21 脚本

(无部件编程)

本章将介绍如何使用 GP-Pro EX 在 “无部件情况下编程” 以及如何创建脚本。
请首先阅读 "21.1 设置菜单 " (p21-2) 然后转到相应页面。

21.1 设置菜单

您可以使用 D 脚本来创建简单的程序。使用该功能，您可以在 GP 中执行操作或在 GP 与不支持的外接设备之间进行通讯。

⚠ 警 告

切勿使用 D 脚本 / 全局 D 脚本来控制可能造成生命威胁或严重伤害的系统。

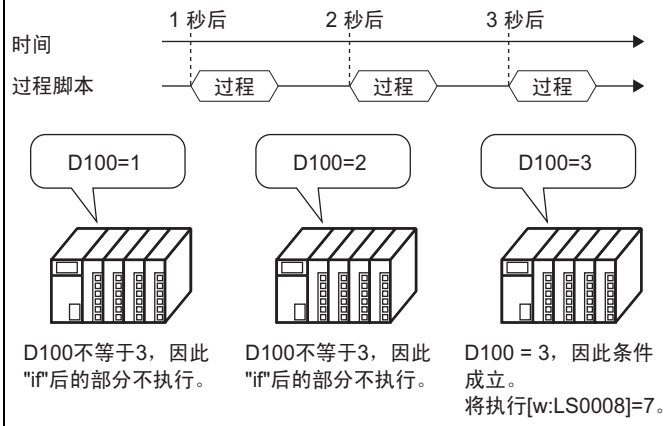
注 释

- D 脚本在基本画面上设置。基本画面在显示时查看条件并执行脚本。
- 当 GP 运行时，全局 D 脚本根据触发而运行，与显示的画面无关。
- 应将扩展脚本用于高级通讯程序。
- 除脚本外，还可以使用逻辑程序来控制应用。

☞ "28.1 设置菜单 " (p28-2)

条件运算

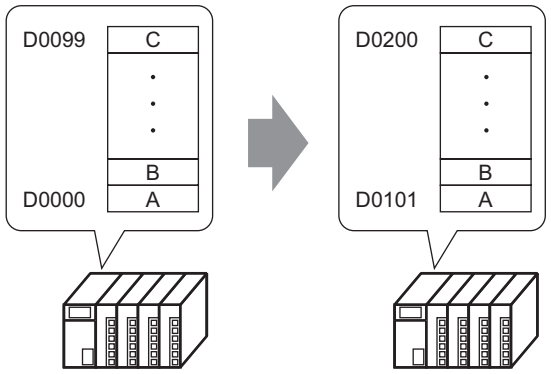
创建一个脚本，在三秒后自动将画面切换为画面编号7。



☞ 设置步骤 (p21-7)
☞ 简介 (p21-6)

复制数据块

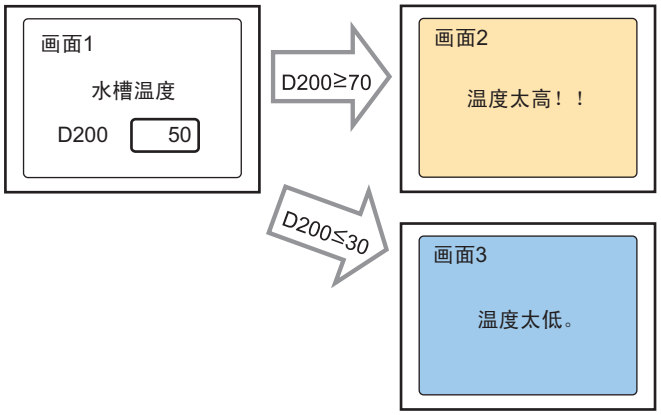
创建一个脚本，用于检测位地址 M100 的上升沿 (0 到 1)，并将保存在外接设备中的数据复制到另一个地址中。



☞ 设置步骤 (p21-13)
☞ 简介 (p21-12)

发生错误时显示报警

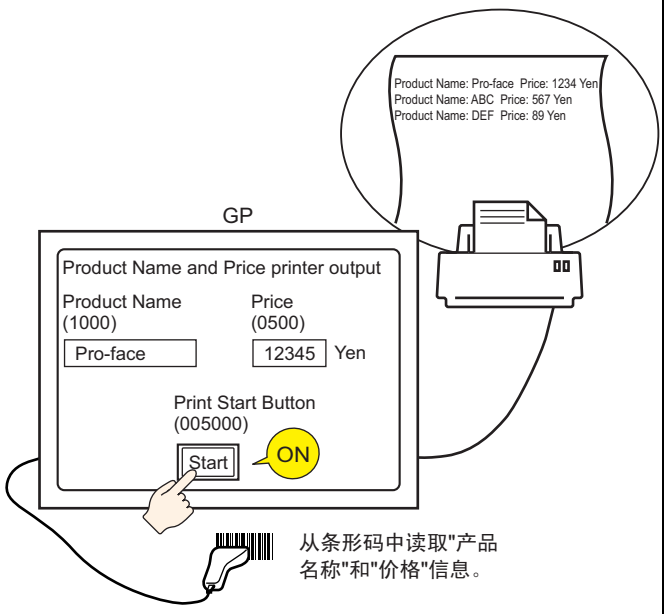
当温度信息存储地址 (D200) 升到 70 度以上或降至 30 度以下时，温度管理系统会检测错误位并显示报警消息。此外，该脚本还对检测到的错误进行计数。



☞ 设置步骤 (p21-18)
☞ 简介 (p21-17)

与不支持的外接设备通讯

创建一个扩展脚本，从连接到 USB 端口的条形码中读取数据并把数据输出至连接到 COM1 的串行打印机上。

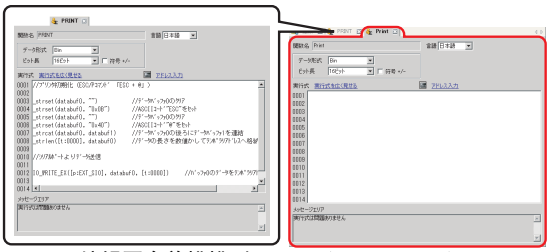


☞ 设置步骤 (p21-34)
☞ 简介 (p21-21)

参考其他脚本

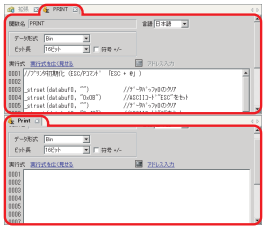
在 [D 脚本] 对话框上，将画面水平或垂直地拆分为两个画面。

可通过点击选项卡切换画面



显示 / 编辑两个并排排列的画面。

水平平铺



垂直平铺

☞ 操作步骤 (p21-39)
☞ 简介 (p21-38)

21.2 条件运算

注 释

- 更多详情，请参阅 “设置指南”。

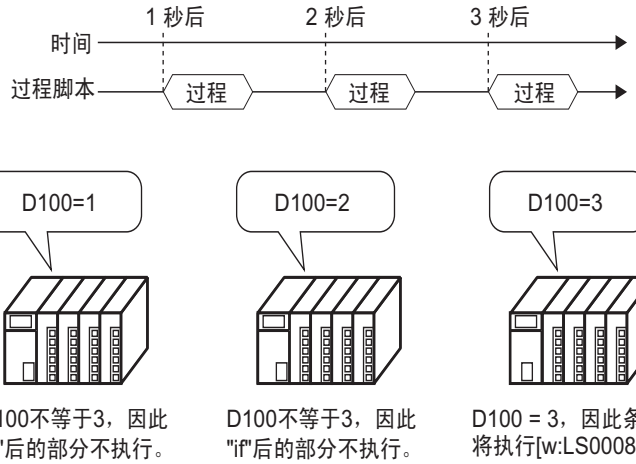
☞"21.9.1 [D 脚本] / [通用设置] - [全局 D 脚本设置] 设置指南 " (p21-52)

- 有关脚本命令的信息，请参阅以下内容。

☞"21.11 指令 / 条件表达式 " (p21-65)

操作

创建一个脚本，在三秒后自动将画面切换为画面编号 7。

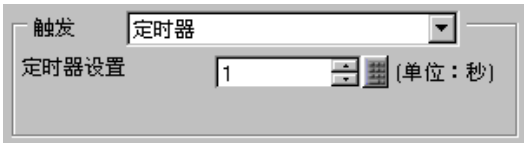


使用的命令

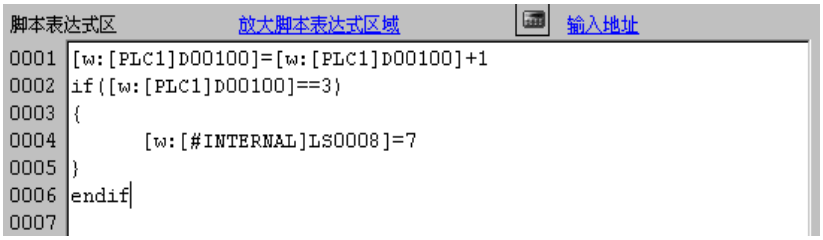
命令	函数摘要
赋值 (=)	把右手侧的值赋给左手侧。 ☞"21.11.10 运算符 " (p21-140)
加 (+)	将一个常量加到字地址数据上。 ☞"21.11.10 运算符 " (p21-140)
if ( )	当条件成立时，执行 “if ( )”语句后面的处理。 ☞"21.11.8 条件表达式 " (p21-134)
等于 (==)	比较右侧和左侧的值。如果右侧值和左侧值相等则条件成立。 ☞"21.11.9 比较 " (p21-138)
LS0008	切换到保存在该值中的画面号。 ☞"A.1.4.2 系统区 " (pA-9)

触发

选择定时器，并将 [定时器设置] 设置为 1 秒。

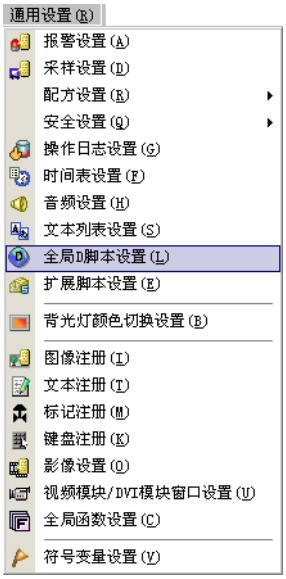


完成的脚本

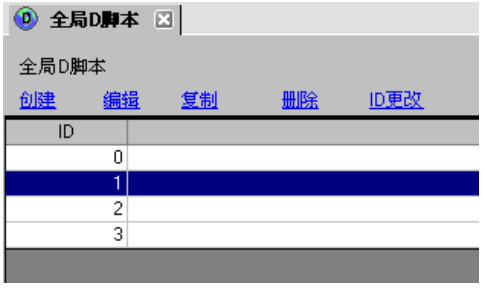


创建步骤

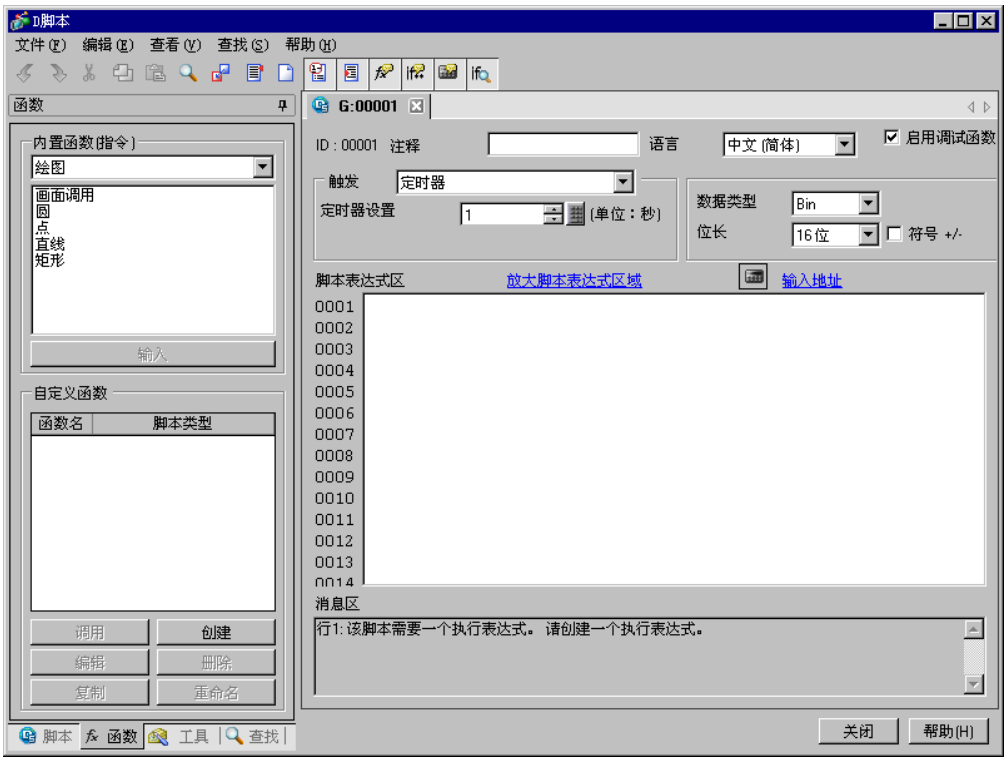
1 在 [通用设置 (R)] 菜单中选择 [全局 D 脚本设置 (L)]。



2 点击 [创建]。要浏览现有脚本，选择 ID 编号并点击 [编辑] 或双击 ID 编号行。

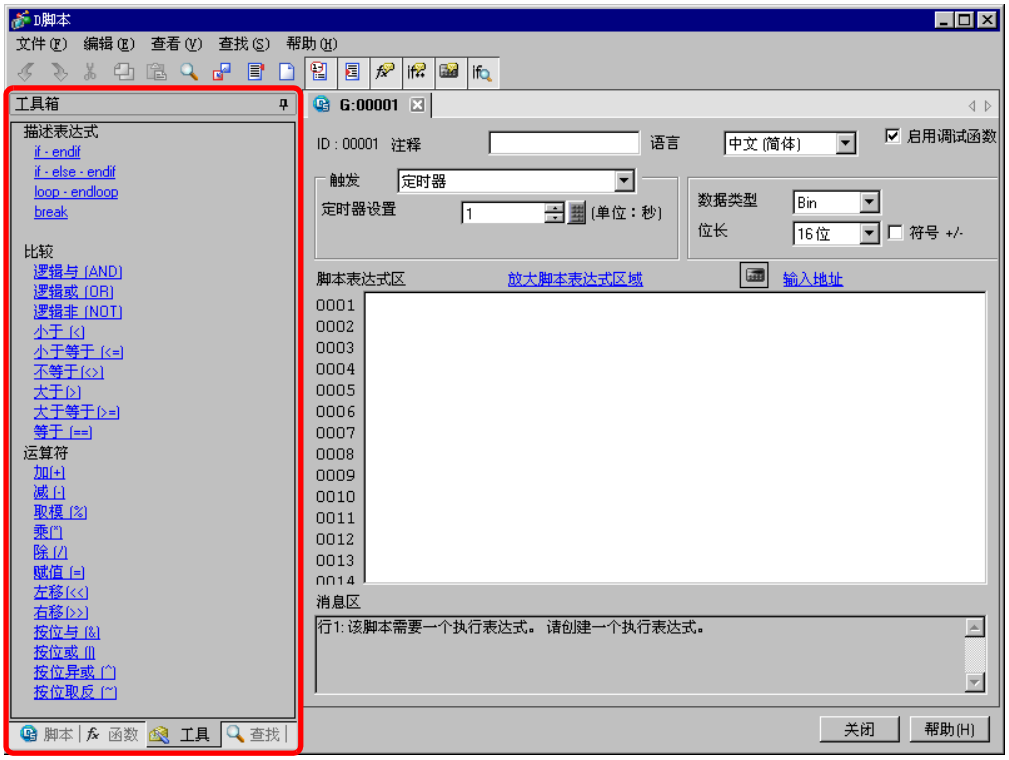


3 然后将显示 [D 脚本] 对话框。

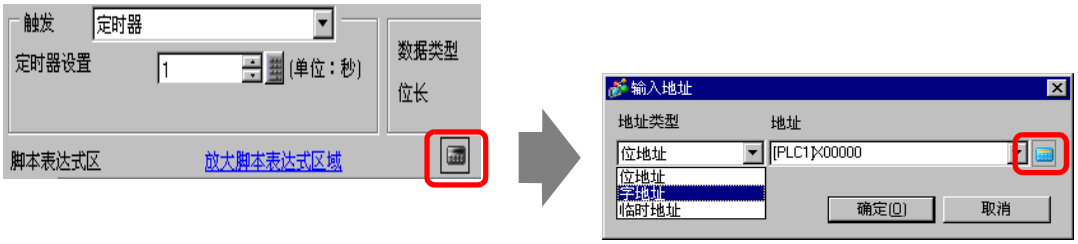


4 在 [触发] 中选择 [定时器] 并指定 [定时器设置] 为 1 秒。

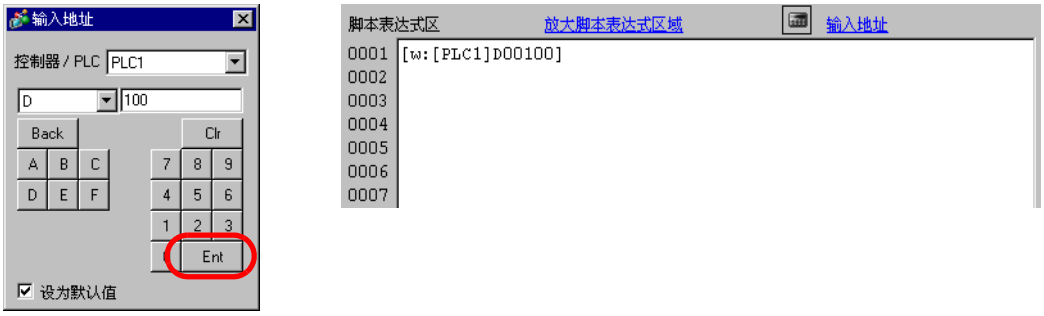
5 点击 [工具箱] 选项卡。工具箱使您能够轻松地放置一条在脚本中使用的命令。



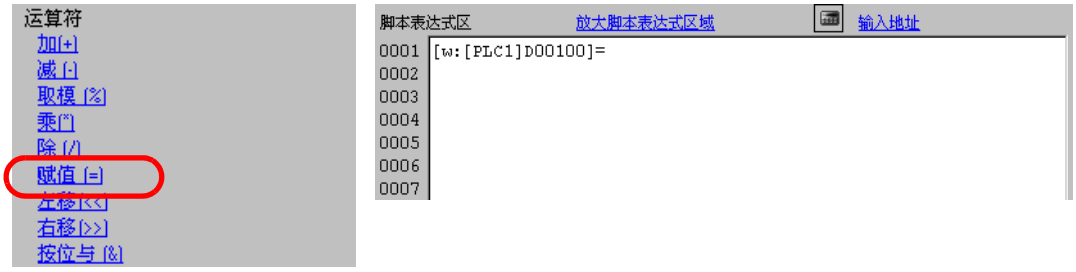
6 创建脚本的第一行。如果将 D00100 的默认值指定为 0，第一行操作就会是一个计数操作，并在每次处理完成时都加 1 并保存。
点击 并选择 [字地址]，点击 。



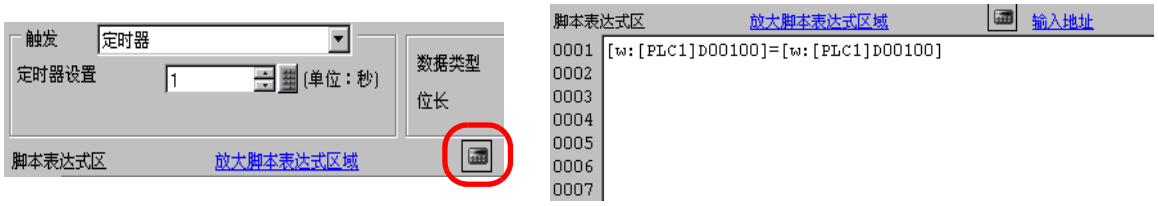
7 输入 D00100，然后点击 [ENT]。在 [地址输入] 对话框中点击 [确定]。



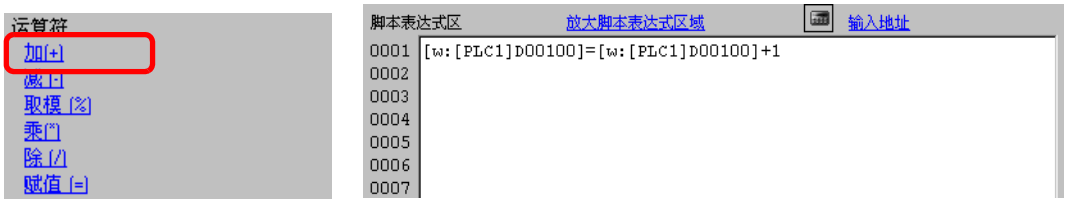
8 点击工具箱中的 [赋值 (=)]。



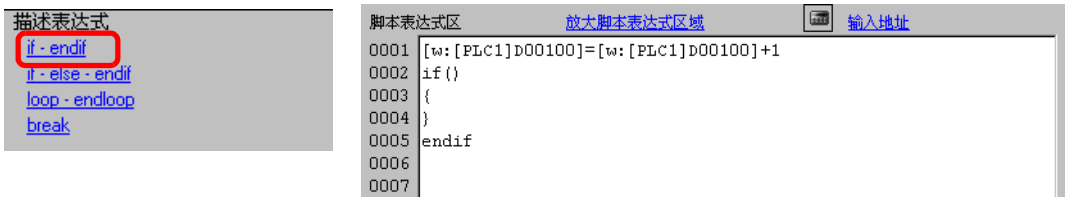
9 按照步骤 6 至 7 中的同样方式放置 D00100。



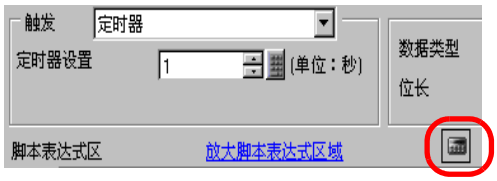
10 点击 [加 (+)] 并键入 “1”。第一行现在完成。

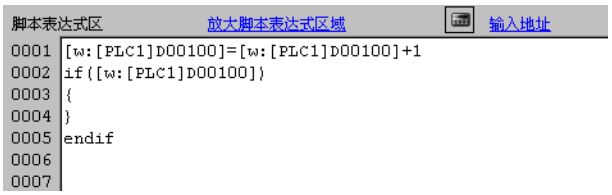


11 创建脚本的第二行。在第二行中，当条件成立时，执行 “if( )”语句后面的处理。点击 [if - endif]。

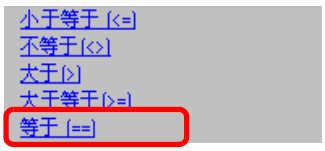


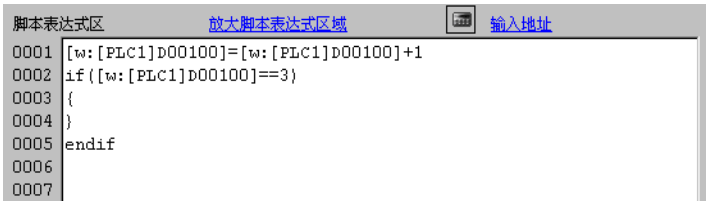
12 在 “if”后面的括号 “( )”内创建条件表达式。条件表达式将保存在 D00100 中的值与 "3" 进行比较，如果相等就为真。将光标放在 “( )”内，重复步骤 6 至 7，放置另外一个 D00100。



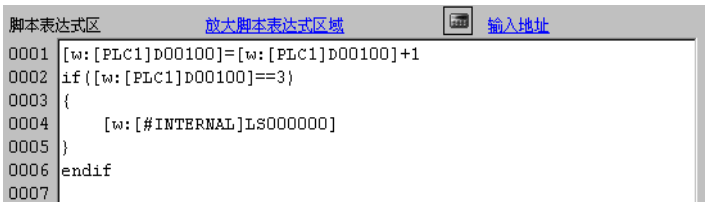


13 点击 [等于 (==)] 并输入 “3”。第二行现在完成。

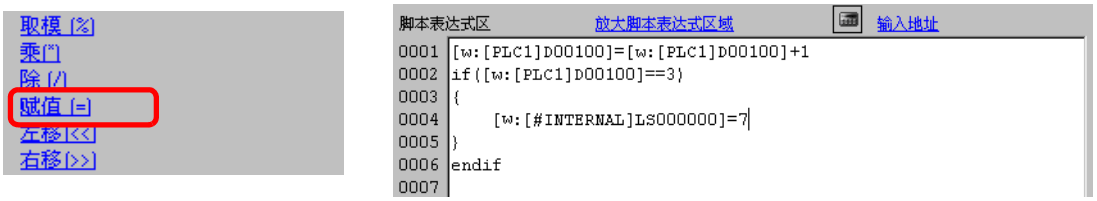




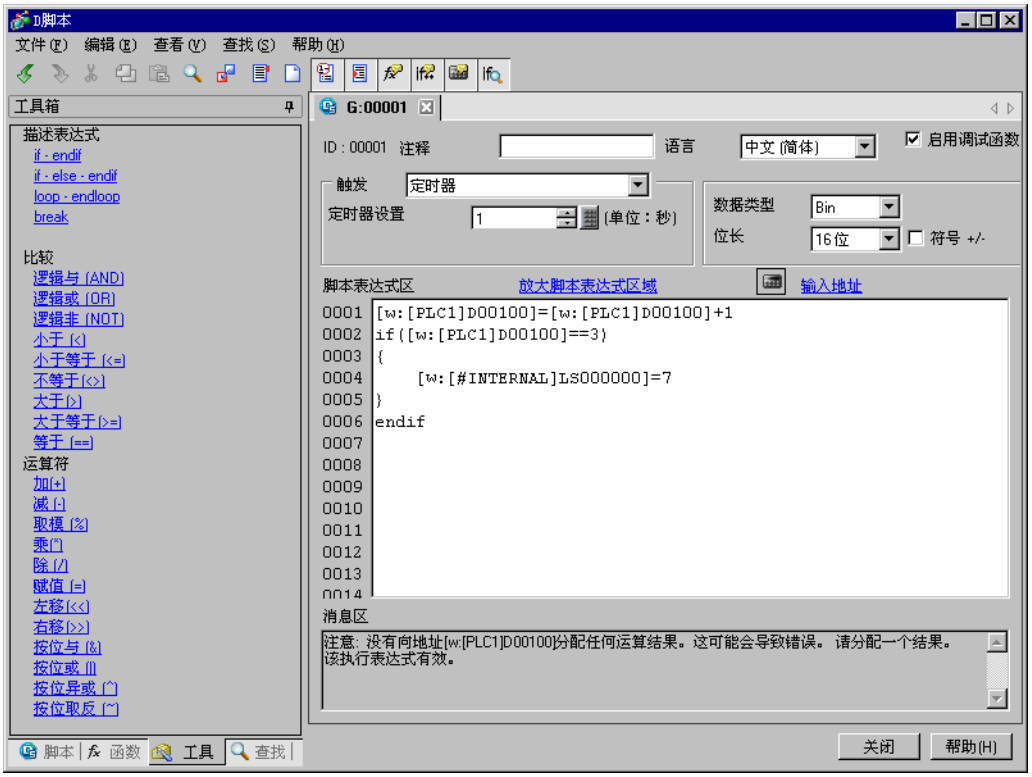
14 将光标放在 “{}”括号内并按 Enter 键。重复步骤 6 至 7 的操作来放置另外一个 LS0008。



15 点击 [赋值 (=)] 并输入 “7”。



16 脚本现在创建完成。



注 释

- 当选择文本时，按 [Ctrl] 键 + [Shift] 键 +[右箭头] 键 /[左箭头] 键来选择整个文本块。
- 按 [Ctrl] 键 + [F4] 键来关闭当前选择的画面。
- 按 [Esc] 键来覆盖并保存脚本或删除脚本并退出。

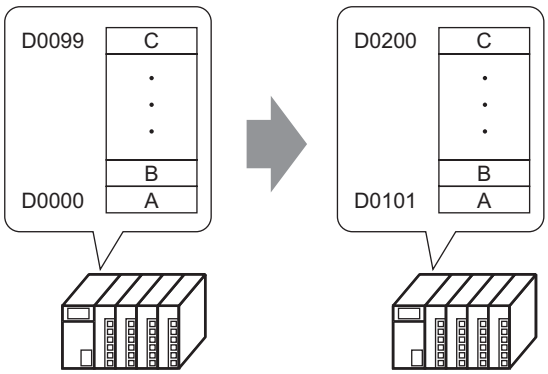
21.3 复制数据块

注 释

- 更多详情，请参阅 “设置指南”。
☞ "21.9.1 [D 脚本] / [通用设置] - [全局 D 脚本设置] 设置指南 " (p21-52)
- 有关脚本命令的信息，请参阅以下内容。
☞ "21.11 指令 / 条件表达式 " (p21-65)

操作

创建一个脚本，用于检测位地址 M100 的上升沿 (0 到 1)，并将保存在外接设备中的数据复制到另一个地址中。

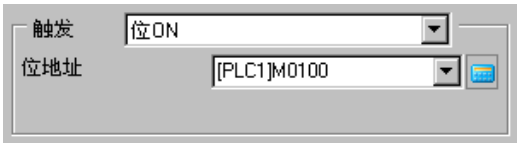


使用的命令

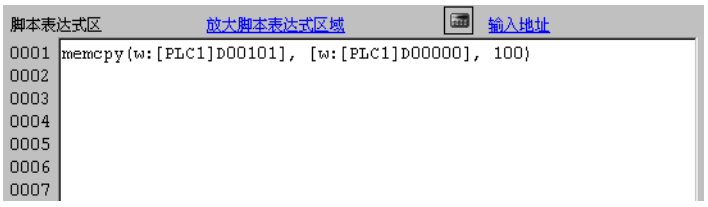
命令	函数摘要
复制存储器 memcpy ( )	在一次操作中将保存的值复制到寄存器。 从源数据的首个字地址开始，将多个地址中的数据复制到目标字地址。 [格式] memcpy ([复制目标地址]， [复制源地址]，字数量) ☞ "21.11.3 存储器操作 " (p21-74)

触发

在 [触发] 中选择 [位 ON]，将 [位地址] 设置为 M000100。

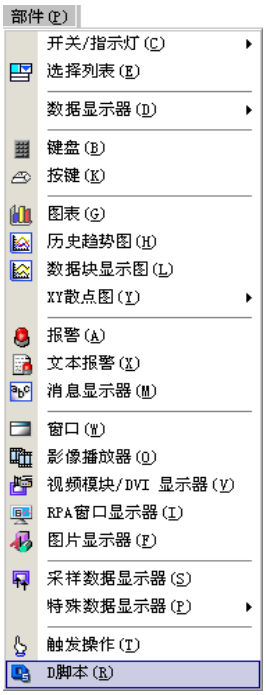


完成的脚本



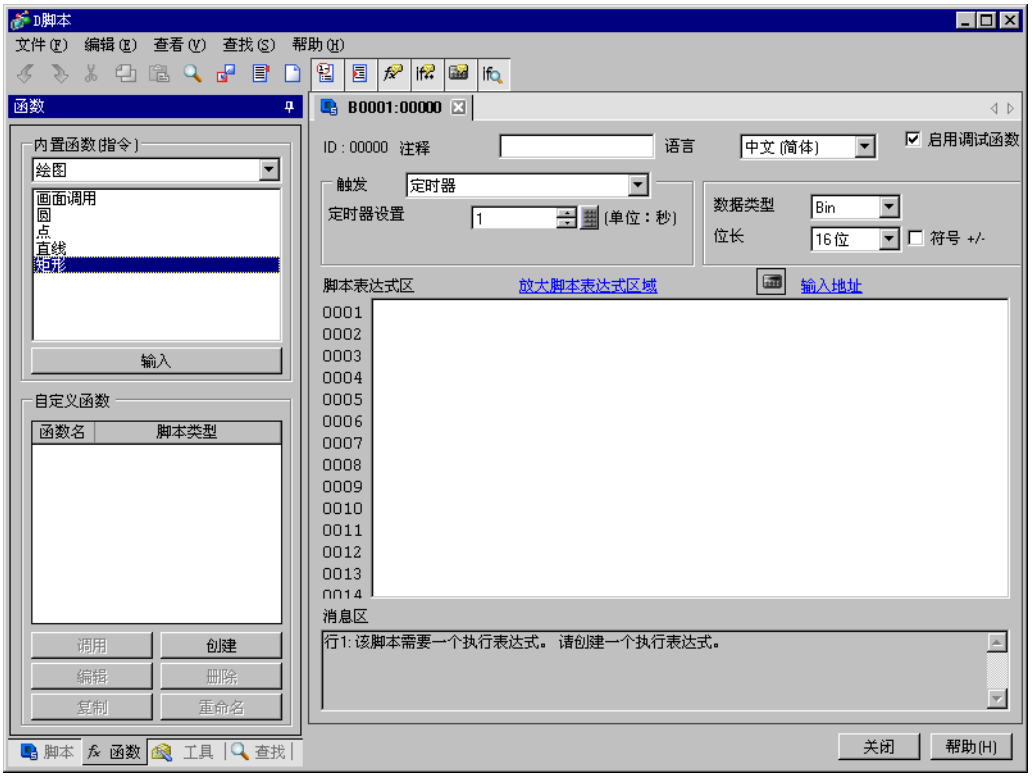
创建步骤

1 在 [部件 (P)] 菜单中选择 [D 脚本 (R)] 或从工具栏中点击  。

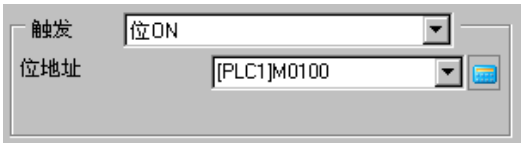


2 点击 [创建]。在 [D 脚本列表] 中将显示现有脚本的 ID。

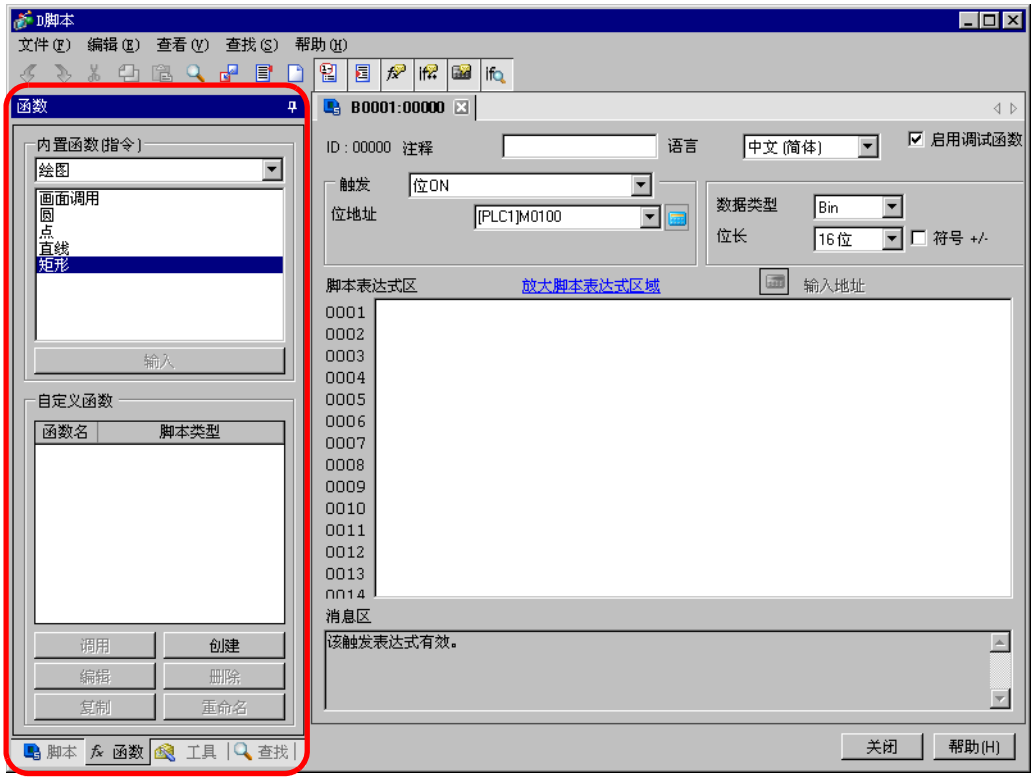
3 然后将显示 [D 脚本] 对话框。



4 在 [触发] 中选择 [位 ON] 并指定 [位地址] 为 M000100。

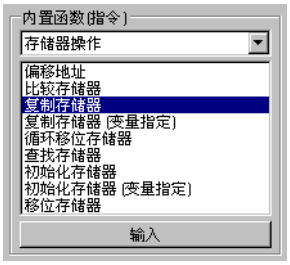


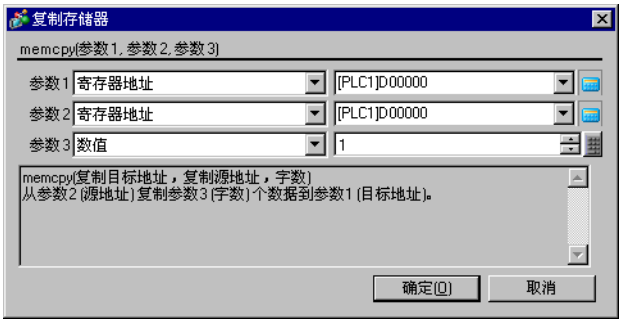
5 点击 [函数] 选项卡。内置函数使您能够轻松地放置一条在脚本中使用的命令。



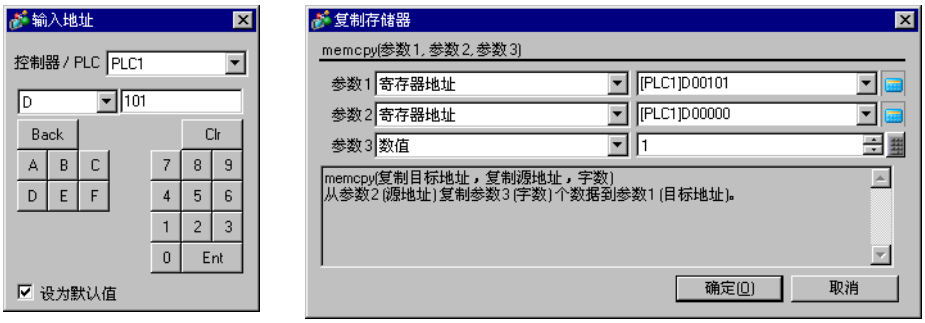
6 从 [内置函数 (指令)] 中选择 [存储器操作]。

7 双击 [复制存储器]，并在接下来的对话框中定义目标地址、源地址和字数等参数。点击  。



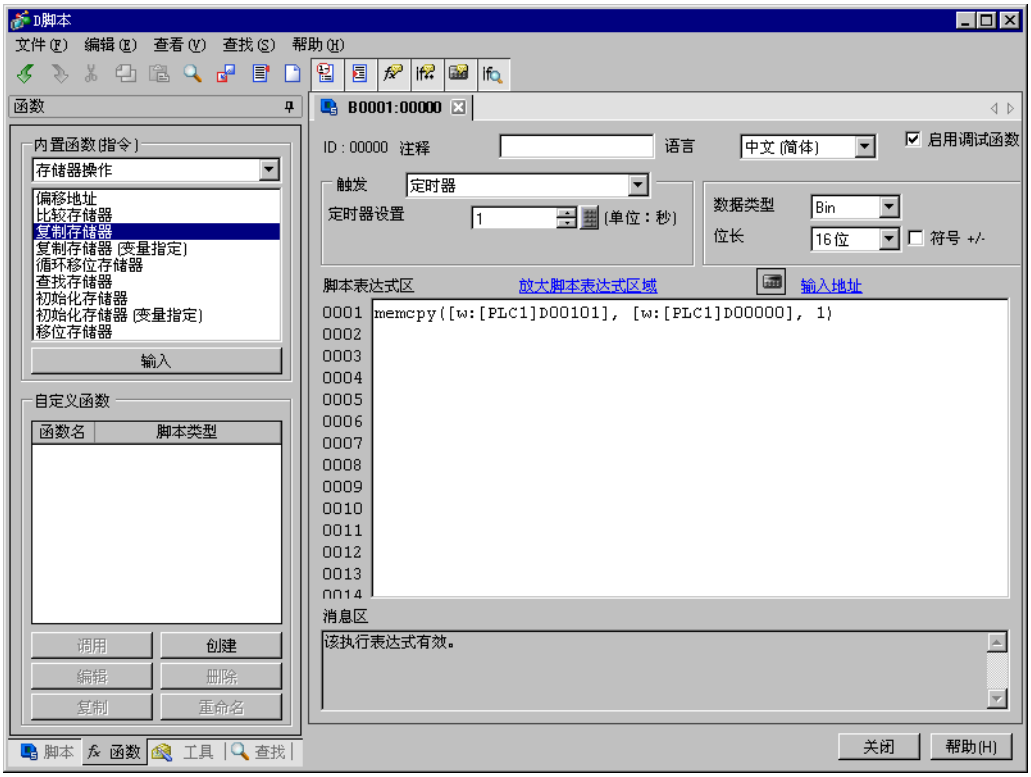


8 在 [参数 1] 中输入 D00101 并点击 [ENT]。



9 在 [参数 2] 中输入 D00000，然后点击 [确定]。

10 脚本现在创建完成。



21.4 发生错误时显示报警

注 释

- 更多详情，请参阅 “设置指南”。

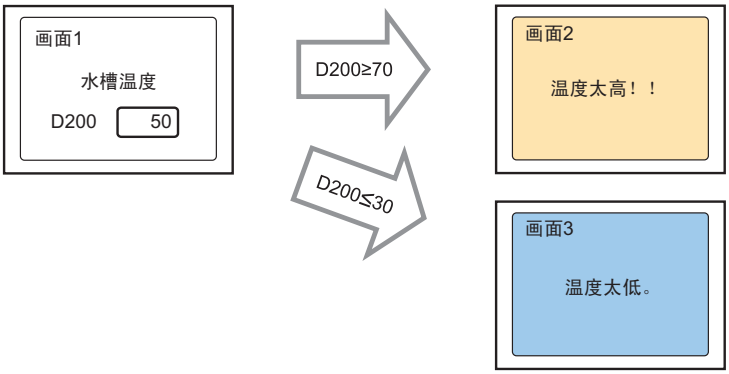
☞"21.9.1 [D 脚本] / [通用设置] - [全局 D 脚本设置] 设置指南 " (p21-52)

- 有关脚本命令的信息，请参阅以下内容。

☞"21.11 指令 / 条件表达式 " (p21-65)

操作

当温度信息存储地址 (D200) 升到 70 度以上或降至 30 度以下时，温度管理系统会检测错误位并显示报警消息。此外，该脚本还对检测到的错误进行计数。



每次当 D200 升高到 70 度以上时进行计数并保存次数的地址：LS0300
每次当 D200 降到 30 度以下时进行计数并保存次数的地址： LS0301
存储报警画面号的地址： LS0008

使用的命令

命令	函数摘要
if ( )	当括号 “( )” 内的 “if” 条件成立时，就运行 “if( )” 语句后的表达式。 ☞"21.11.8 条件表达式 " (p21-134)
大于等于 (>=)	如果 N1 大于等于 N2(N1>=N2) 时成立。 ☞"21.11.9 比较 " (p21-138)
赋值 (=)	把右手侧的值赋给左手侧。 ☞"21.11.10 运算符 " (p21-140)
加 (+)	将一个常量加到字地址数据上。 ☞"21.11.10 运算符 " (p21-140)
小于等于 (<=)	N1 小于等于 N2(N1<=N2) 时成立。 ☞"21.11.9 比较 " (p21-138)

触发

在 [触发] 中选择 [位 ON]，将 [位地址] 设置为 M000100。

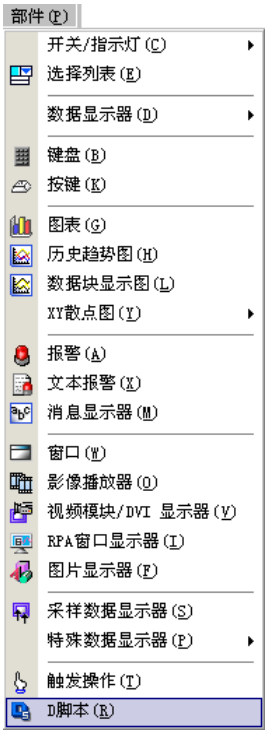
完成的脚本

脚本表达式区　放大脚本表达式区域　输入地址

```
0001 if([w:[PLC1]D00200]>=70)                              //When temp is greater than 70 degrees
0002 {
0003     [w:[#INTERNAL]LS0302]=100                         //Greater than 70 degrees alarm screen number 100
0004     [w:[#INTERNAL]LS0300]=[w:[#INTERNAL]LS0300]+1     //Increase error count
0005 }
0006 endif
0007
0008 if([w:[PLC1]D00200]>=30)                              //When temp is greater than 30 degrees
0009 {
0010     [w:[#INTERNAL]LS0302]=101                         //Greater than 30 degrees alarm screen number 101
0011     [w:[#INTERNAL]LS0301]=[w:[#INTERNAL]LS0301]+1     //Increase error count
0012 }
0013 endif
0014
```

创建步骤

1 从 [部件] 菜单中点击 [D 脚本 (R)] 或点击  。



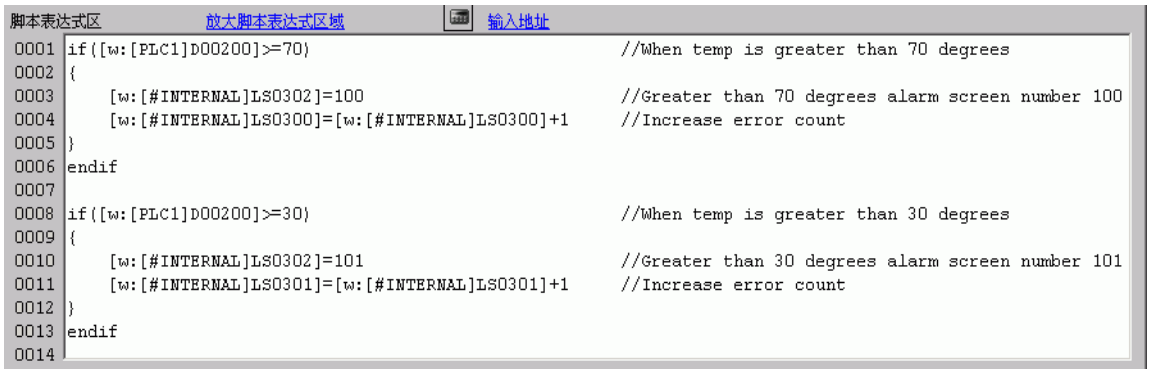
2 点击 [创建]。在 [D 脚本列表] 中将显示现有脚本的 ID。

3 然后将显示 [D 脚本] 对话框。

4 设置注释。输入 “报警显示器”。

5 在 [触发] 中选择 [位 ON]，将 [位地址] 设置为 M000100。

6 通过向脚本表达区添加函数、语句和表达式来创建程序，完成脚本创建。



注 释

- 当选择文本时，按 [Ctrl] 键 + [Shift] 键 +[右箭头] 键 /[左箭头] 键来选择整个文本块。
- 按 [Ctrl] 键 + [F4] 键来关闭当前选择的画面。
- 按 [Esc] 键来覆盖并保存脚本或删除脚本并退出。

21.5 与不支持的外接设备通讯

注 释

- 更多详情，请参阅 “设置指南”。

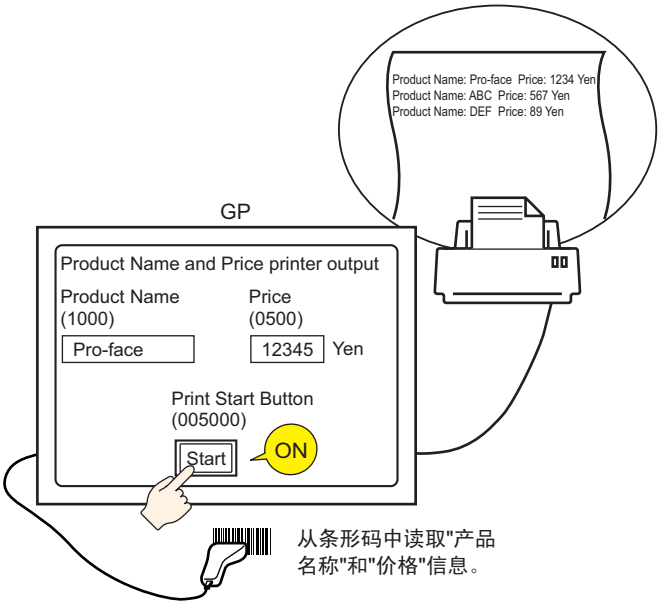
☞"21.9.1 [D 脚本] / [通用设置] - [全局 D 脚本设置] 设置指南 " (p21-52)

- 有关脚本命令的信息，请参阅以下内容。

☞"21.11 指令 / 条件表达式 " (p21-65)

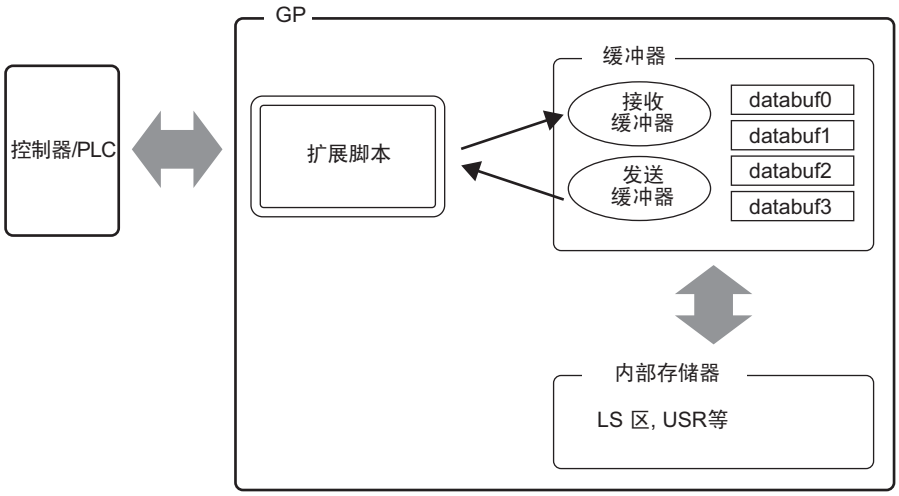
■ 操作设置

创建一个扩展脚本，从连接到 USB 端口的条形码阅读器中读取数据并将数据输出至连接到 COM1 的串行打印机上。



■ 扩展脚本的结构

扩展脚本用于 GP 内部串口和所连接的输入 / 输出设备之间的通讯。
关于扩展脚本数据管理，如下图所示，通过发送 / 接收缓冲器将数据保存在数据缓冲器 databuf0 至数据缓冲器 databuf3 中。数据缓冲器不是按地址区分的，因此在编辑控制器 /PLC 上的数据前应在内存中保存数据。



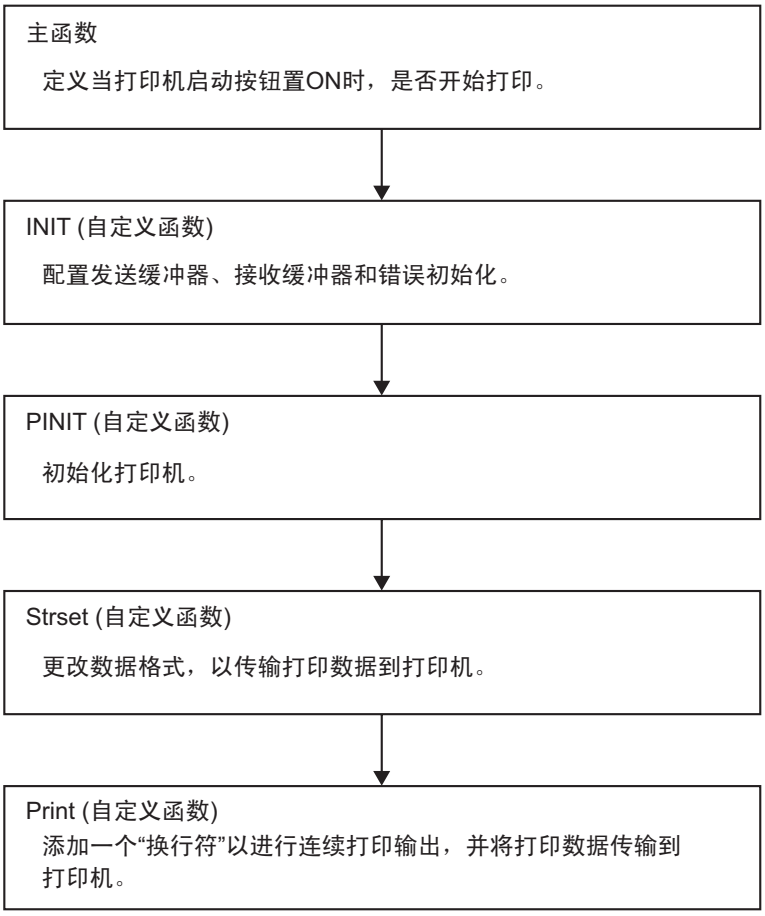
接收缓冲器 / 发送缓冲器

对于与控制器 /PLC 的通讯，它相当于一个位存储空间，可实时识别发送和接收的数据。

databuf0 - databuf3

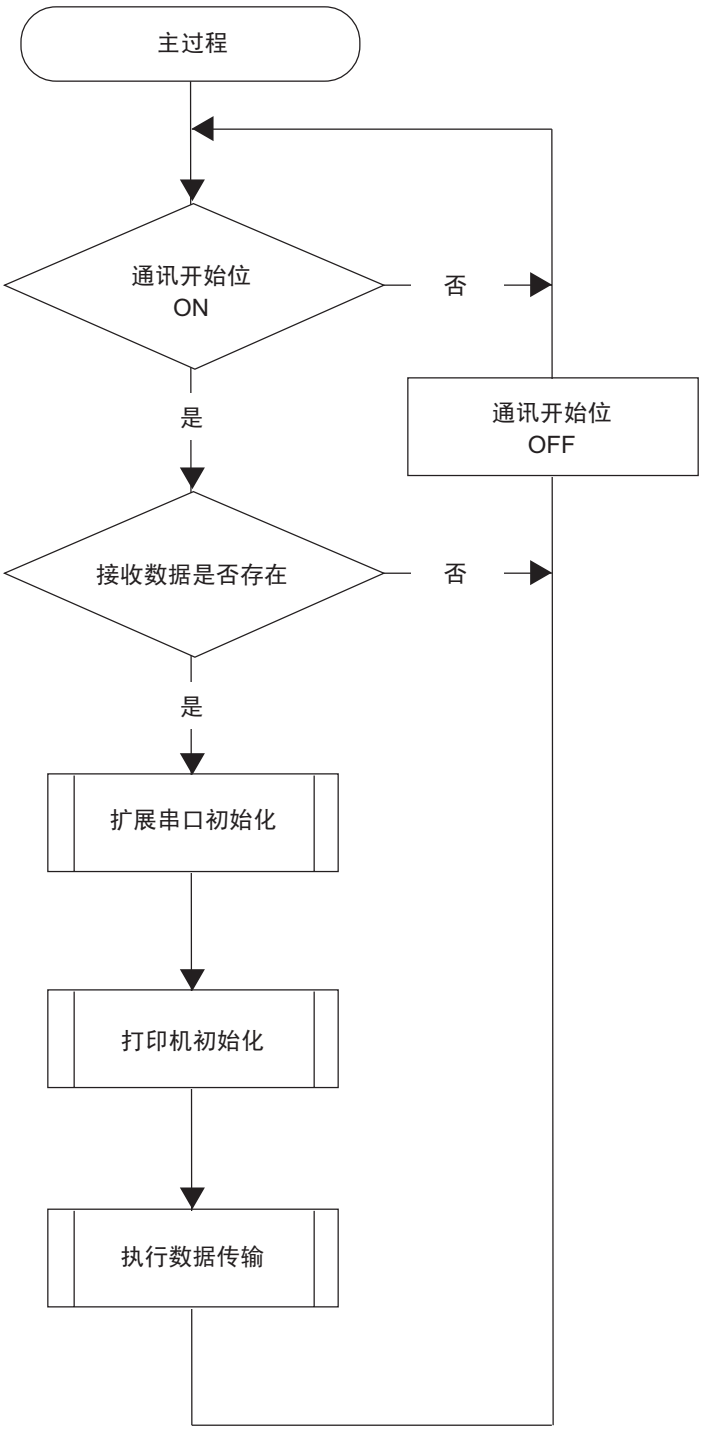
这些是用于数据存储的字节 (8 位) 存储空间。缓冲器大小是 1KB。

■ 创建脚本步骤

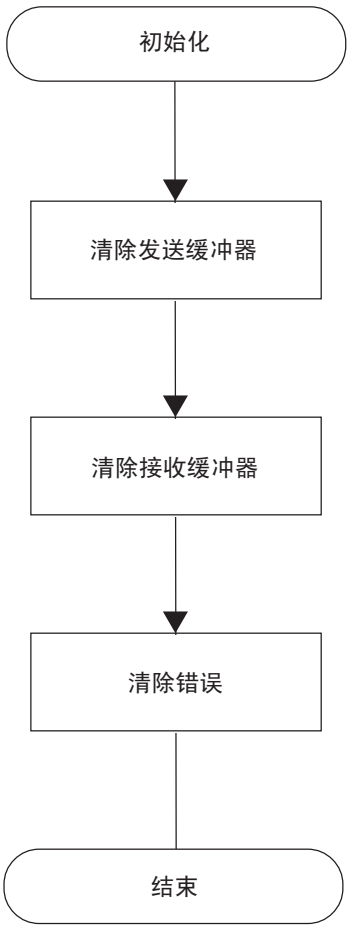

主函数
定义当打印机启动按钮置ON时，是否开始打印。
INIT (自定义函数)
配置发送缓冲器、接收缓冲器和错误初始化。
PINIT (自定义函数)
初始化打印机。
Strset (自定义函数)
更改数据格式，以传输打印数据到打印机。
Print (自定义函数)
添加一个“换行符”以进行连续打印输出，并将打印数据传输到打印机。

■ 流程图

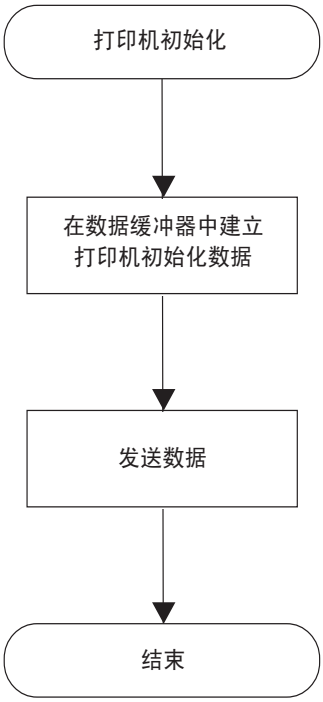
(1) 主过程



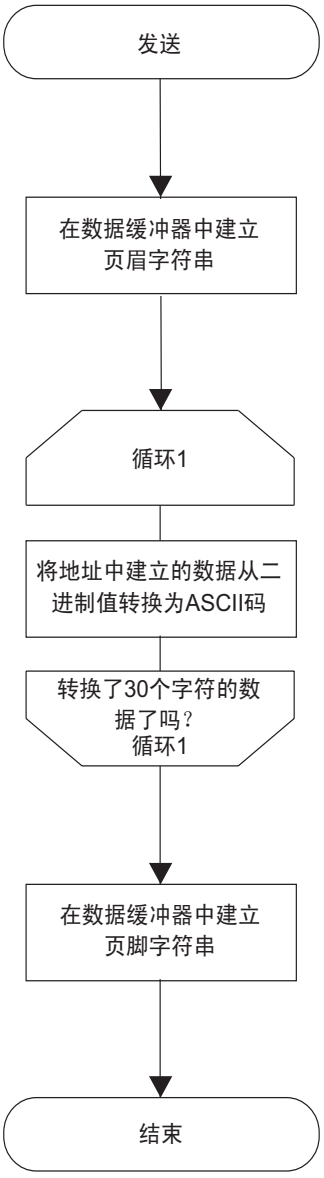
(2) 初始化函数 (INT)



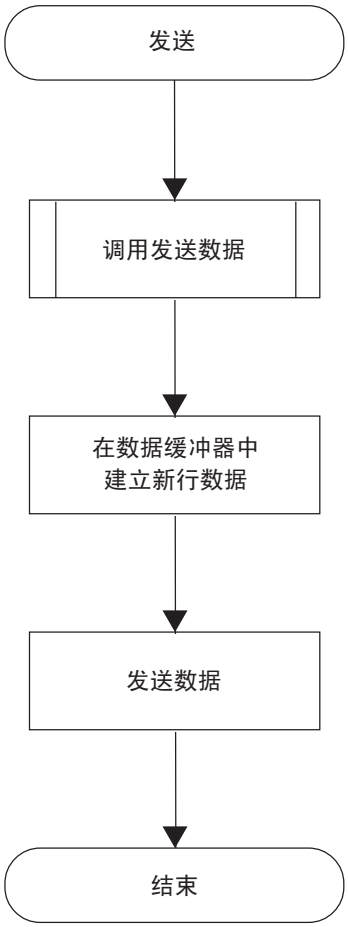
(3) 打印机初始化函数 (PINIT)



(4) 字符串函数 (Strset)



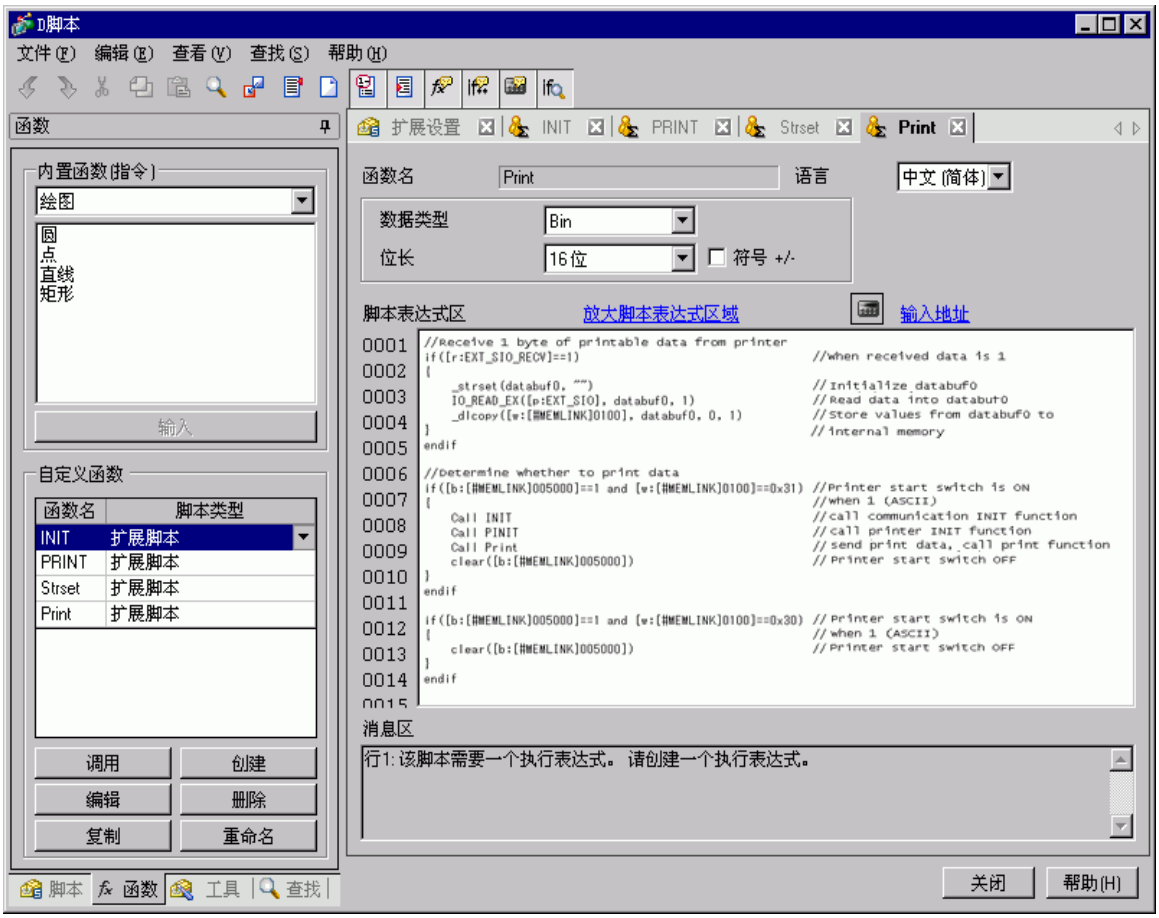
(5) 发送函数 (Print)



■ 脚本操作概述

◆ 主函数

完成的脚本



函数摘要

当打印机启动按钮 (内部存储器 00D5000) 置 ON 时，脚本决定是否从打印允许数据的第一个字节开始打印。

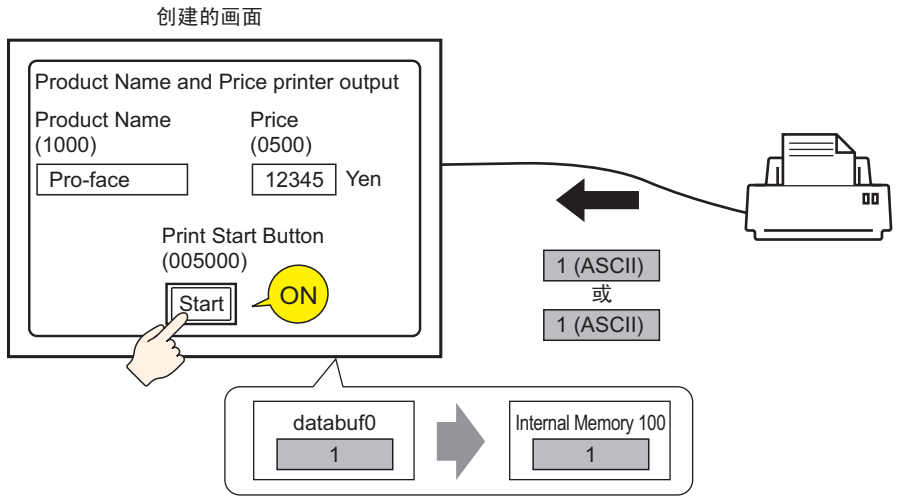
作为打印机指定的一个示例，打印许可数据执行下述操作。

打印准备就绪：向控制器 /PLC 发送 0x31(ASCII 代码 “1”)。

打印准备无效：向控制器 /PLC 发送 0x30(ASCII 代码 “0”)。

GP 在数据缓冲器 0 中接收打印许可数据，在随后的脚本处理中该数据被移动到可访问的内部存储器 100 中。

当内部存储器 100 = 0x31(值 “1” 的 ASCII 代码) 时，打印开始。当内部存储器是 0x30(“0” 的 ASCII 代码) 时， GP 返回脚本开始的地方并重复该过程，直到收到 0x31 数据。



◆ INIT(自定义函数)

完成的脚本

脚本表达式区　放大脚本表达式区域　输入地址

```
0001 [c:EXT_SIO_CTRL00]=1    //Send buffer clear
0002 [c:EXT_SIO_CTRL01]=1    //Receive buffer clear
0003 [c:EXT_SIO_CTRL02]=1    //Error buffer clear
0004
```

函数摘要

配置发送缓冲器、接收缓冲器和错误初始化。

◆ PINIT(自定义函数)

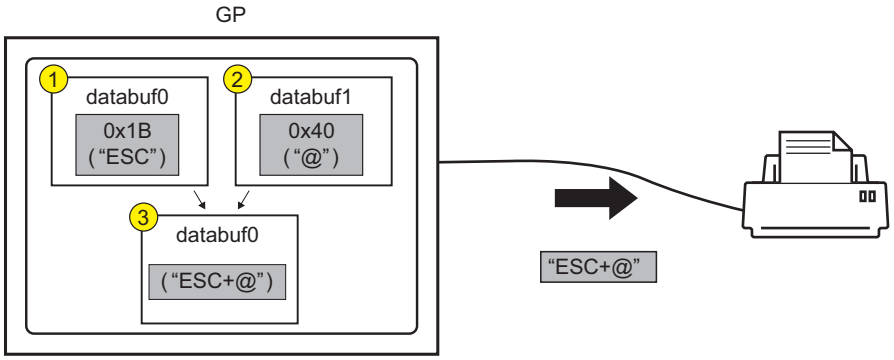
完成的脚本

脚本表达式区　放大脚本表达式区域　输入地址

```
0001 Call Strset                  //call Printing Data function
0002 _strset(databuf0,"")         //clear databuf0
0003
0004 //Printing and Delimiter (Carriage Return / Line Feed)
0005
0006 _strset(databuf0,0x0d)       //Print out ,then go back to first place of the line
0007 _strcat(databuf1,databuf0)   //append databuf0 into the end of databuf1
0008 _strset(databuf0,"")         //clear databuf0
0009 _strset(databuf0,0x0a)       //go to next line
0010 _strcat(databuf1,databuf0)   //append databuf0  into the end of  databuf1
0011
0012 _strlen([t:0000],databuf1)   //store data length to temporary address
0013
0014 //Send data over serial port
0015
0016 IO_WRITE_EX([p:EXT_SIO],databuf1,[t:0000])   //Send databuf0, amount defined by temporary address value
0017
```

函数摘要

初始化打印机。向打印机发送 ESC/P 命令 “ESC+@”。



◆ Strset(自定义函数)

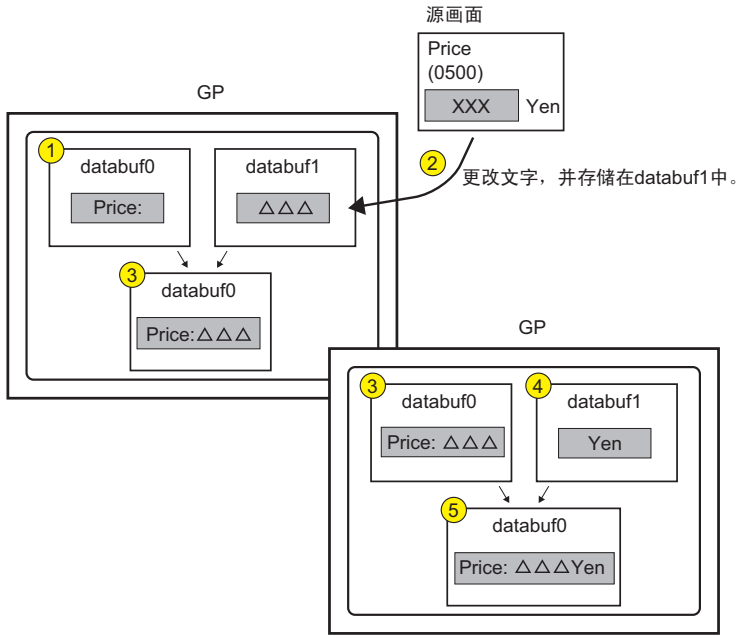
完成的脚本

脚本表达式区 放大脚本表达式区域 输入地址

```
0001 //String example, add "Price:" and "$"
0002 _strset(databuf0, "")                          //Initialize databuf0
0003 _strset(databuf0, "Price:")                    //Store text "Price:" to databuf0
0004 _bin2decasc(databuf0,[w:[#MEMLINK]0500])       //Convert value to string and store in databuf1
0005 _strcat(databuf0, databuf1)                    //Add databuf1 to end of databuf0
0006 _strset(databuf1, "")                          //Initialize databuf1
0007 _strset(databuf1, "$")                         //Store text "$" to databuf1
0008 _strcat(databuf0, databuf1)                    //Add databuf1 to end of databuf0
0009
0010 //Initialize temporary address
0011 [t:0001]=0
0012 [t:0002]=0
0013
0014 //Store to internal memory word units, consecutive characters into byte units(30 characters)
0015 loop()
0016 {
0017  [w:[#MEMLINK]2000]#[t:0002]=[w:[#MEMLINK]1000]#[t:0001]>>8      //Store top byte into bottom byte
0018  [w:[#MEMLINK]2001]#[t:0002]=[w:[#MEMLINK]1000]#[t:0001]& 0*FF   //Erase top byte and store in next address
0019  [t:0001]=[t:0001]+1                                            //Address offset + 1
0020  [t:0002]=[t:0002]+2                                            //Address offset + 2
0021  if([t:0001]==15)                                               //Store 2 words into 2 byte and repeat 15 times
0022  {
0023   break
0024  }
0025  endif
0026 }
0027 endloop
0028 _ldcopy(databuf2, [w:[#MEMLINK]2000],30)        //Store internal memory 2000~2030 to data buffer as characters
0029
0030 //Add string "Item:"
0031 _strset(databuf1, "")                           //Initialize databuf1
0032 _strset(databuf1, "Item:")                      //Store string "Item:" into databuf1
0033 _strcat(databuf1, databuf2)                     //Add databuf1 to end of databuf0
0034
0035 //Add Item and Price strings
0036 _strcat(databuf1, databuf0)                     //Add databuf0 to end of databuf1
```

函数摘要

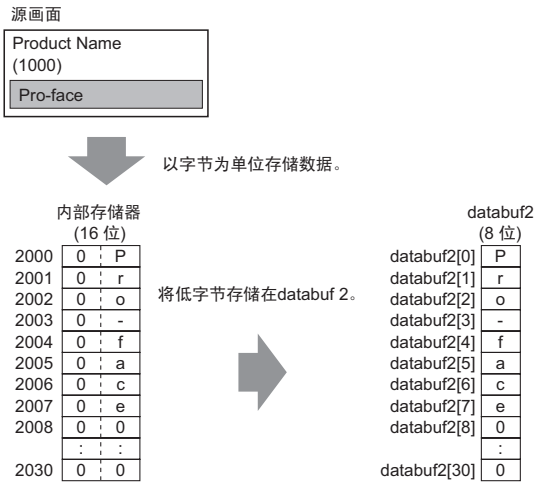
1 将文本 “价格：”和 “日元”附加在内存 0500 保存的价格数据之后。



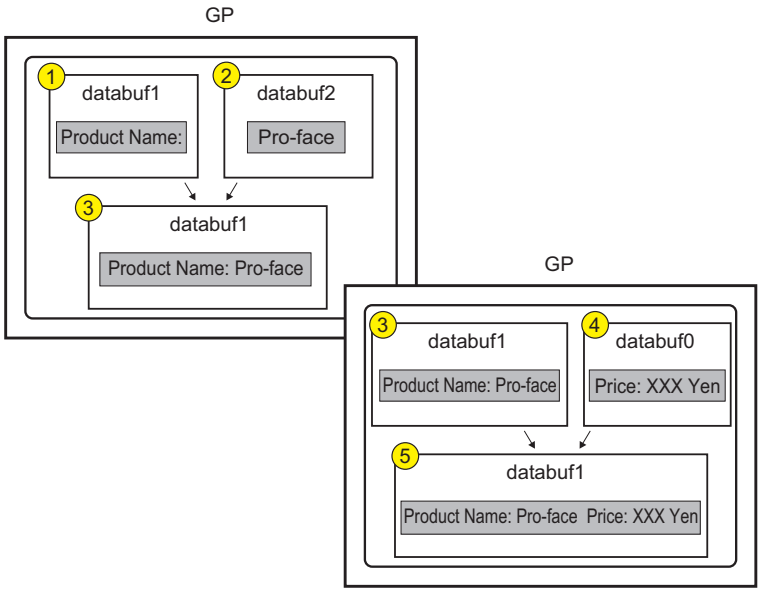
2 更改数据格式以便向打印机发送打印数据。将内部存储器 1000 中顺序保存的字符串数据 (产品名称) 分成字节单位，并作为低字节字符串数据保存在内部存储器 2000 至 2030 中。使用函数 _ldcopy，并按连续字地址的最低字节顺序，将数据保存在 databuf2 中。

注 释

- _ldcopy 函数将取出按字保存的数据，且只将低字节数据保存在缓冲器中，而忽略那些高字节数据。



3 将文本 “产品名称：”和 “价格”加入数据缓冲器 2 中。



◆ **打印 (自定义函数)**

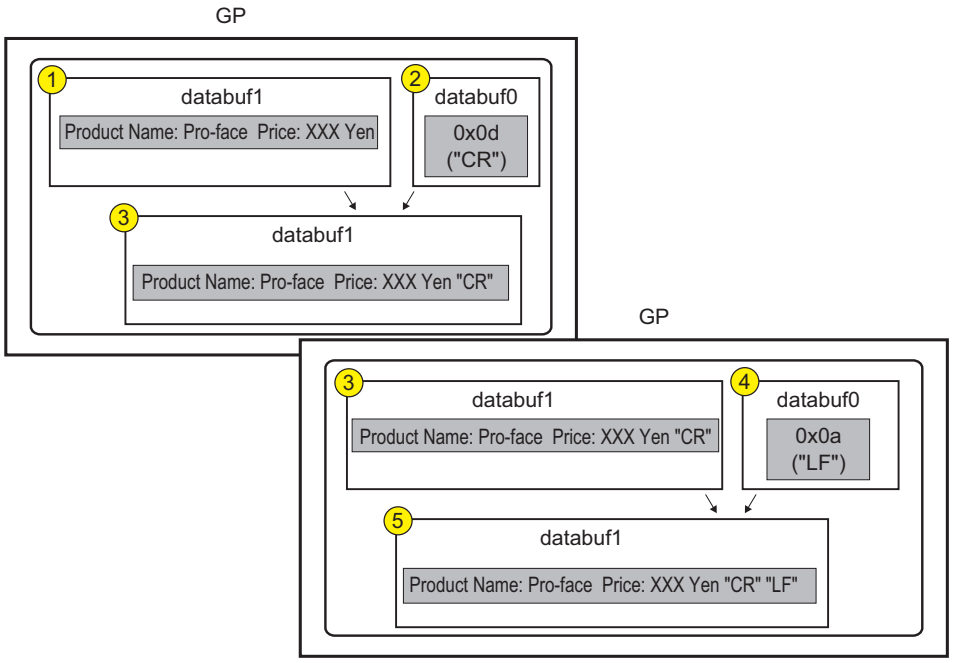
完成的脚本

脚本表达式区 放大脚本表达式区域 输入地址

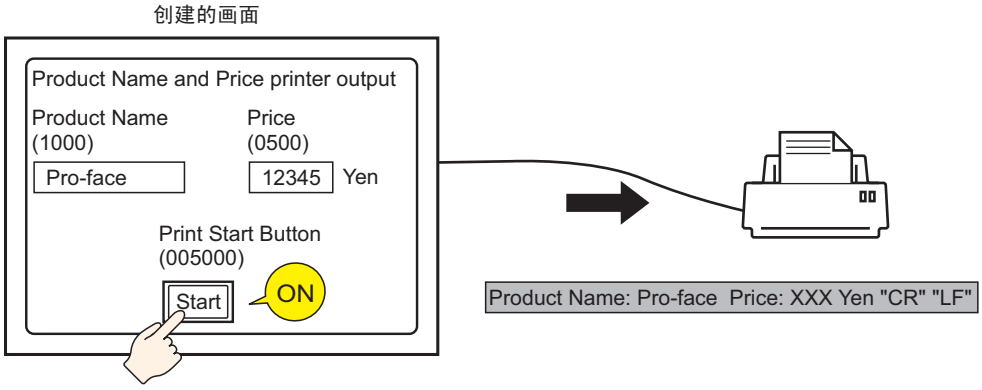
```
0001 Call Street                        //Call string data function
0002 _strset(databuf0,"")               //Clear databuf1
0003
0004 //Text delimiter
0005
0006 _strset(detabuf0, 0*0d)            //Return to start of row
0007 _strset(databuf1, databuf0)        //Add databuf1 to end of databuf0
0008 _strset(databuf0, "")              //Clear databuf1
0009 _strset(databuf0, 0*0a)            //New line
0010 _strset(databuf1, databuf0)        //Add databuf1 to end of databuf0
0011
0012 _strset([t:0000], databuf1)        //Store data length to temporary address
0013
0014 //Send data over serial port
0015
0016 IO_WRITE_EX([p:EXT_SIO], databuf1, [t:0000] //Send databuf0, amount defined by temporary address value
0017
```

函数摘要

1 附加 “换行”以实现连续的打印机输出。



2 向打印机发送数据。

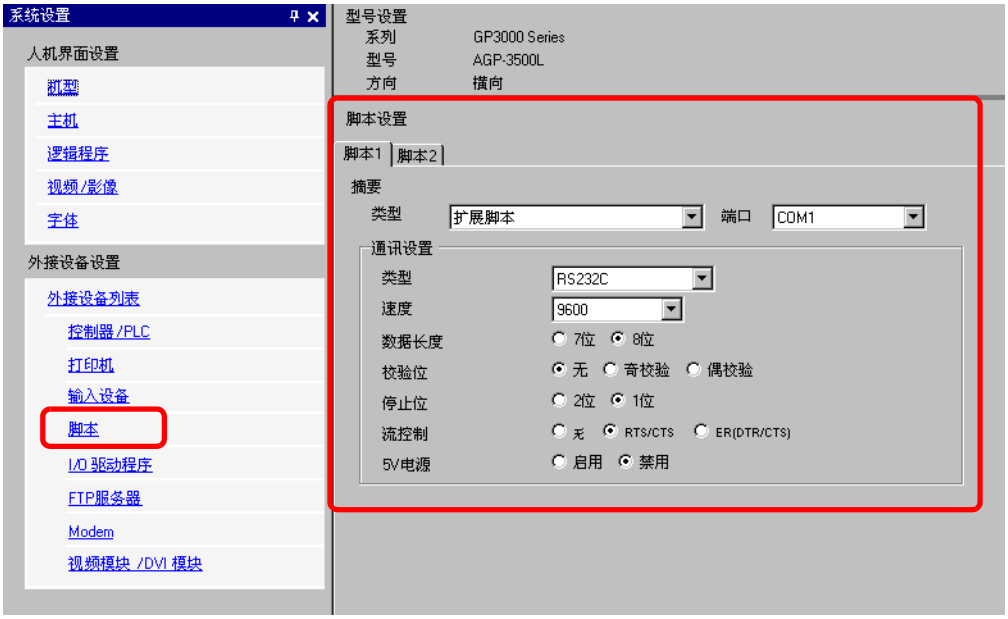


■ 使用的命令

命令	函数摘要
if ( )	当括号 “( )” 内的 “if” 条件成立时，就运行 “if( )” 语句后的表达式。 ☞"21.11.8 条件表达式 " (p21-134)
标签设置 [r:EXT_SIO_RECV]	显示此时已经接收的数据量 (字节数)。已接收数据大小为只读。 ☞"21.11.4 串口操作 " (p21-91)
等于 (==)	N1 等于 N2(N1=N2) 时成立。 ☞"21.11.9 比较 " (p21-138)
文本设置 (_strset)	固定字符串保存在数据缓冲器中。 ☞"21.11.11 文本操作 " (p21-144)
扩展接收 (IO_READ_EX)	从扩展串口中接收数据，数据量在接收数据大小 (字节) 中指定，并将它保存在数据缓冲器中。 ☞"21.11.4 串口操作 " (p21-91)
从数据缓冲器到内部寄存器 (_dlcopy)	根据字符串数将保存在数据缓冲器偏移中的字符串数据的每个字节复制到 LS 区。 ☞"21.11.11 文本操作 " (p21-144)
标签设置 [c:EXT_SIO_CTRL**]	该控制变量用于清除发送缓冲器、接收缓冲器和错误状态。 ☞"21.11.8 条件表达式 " (p21-134)
连接文本 (_strcat)	用文本缓冲器将字符串或字符代码连接在一起。 ☞"21.11.11 文本操作 " (p21-144)
文本长度 (_strlen)	获取已保存字符串的长度。 ☞"21.11.11 文本操作 " (p21-144)
扩展发送 (IO_WRITE_EX)	根据发送字节数大小用扩展串口发送数据缓冲器中的数据。 ☞"21.11.4 串口操作 " (p21-91)
赋值 (=)	把右手侧的值赋给左手侧。 ☞"21.11.10 运算符 " (p21-140)
加 (+)	将一个常量加到字地址数据上。 ☞"21.11.10 运算符 " (p21-140)
数值到十进制字符串转换 (_bin2decasc)	该函数用于将整数转换成十进制字符串。 ☞"21.11.11 文本操作 " (p21-144)
从内部寄存器到数据缓冲器 (_ ldcopy)	根据字符串数，将保存在 LS 区的字符串数据以逐字节传输的方式复制到数据缓冲器。 ☞"21.11.11 文本操作 " (p21-144)

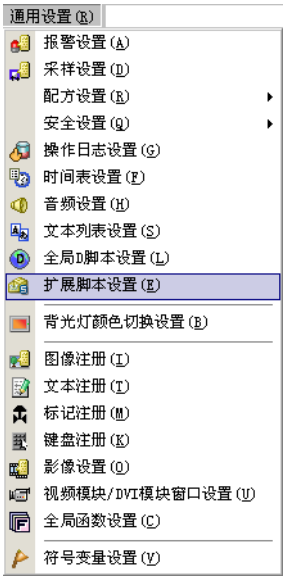
■ 创建步骤

1 进行脚本设置，使用扩展脚本进行通讯。从[工程(F)]菜单中点击[系统设置(C)]。选择[脚本]。将[类型]设置为[扩展脚本]。



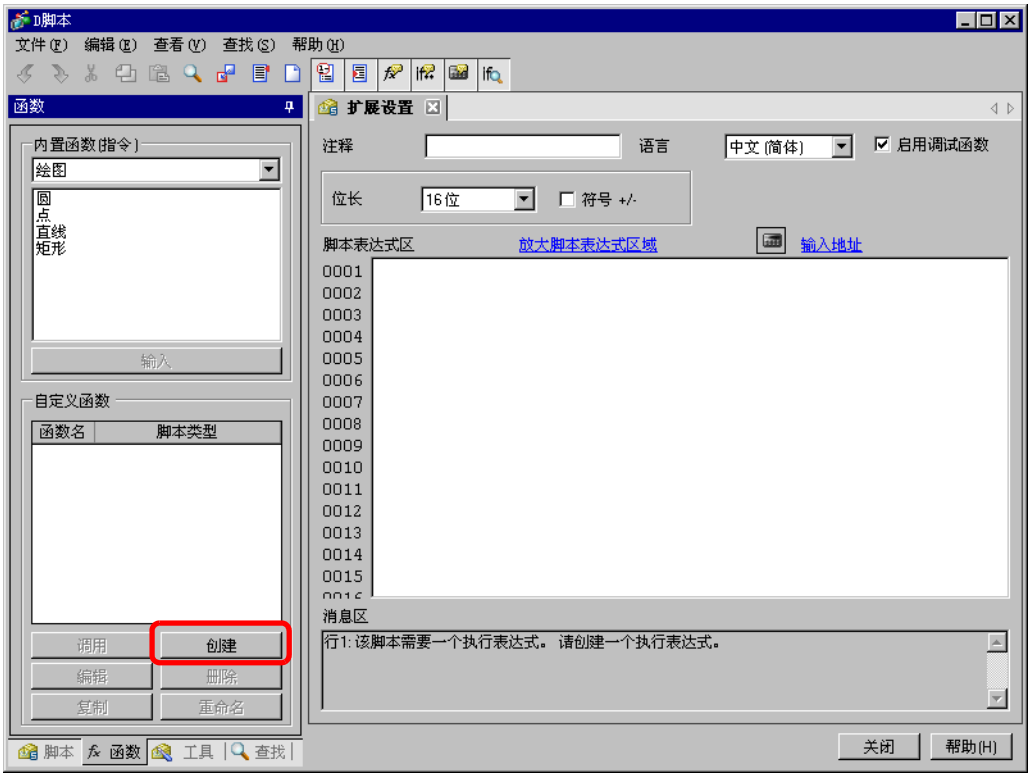
有两个标签可以设置脚本。将[端口]设置为 COM1 或 COM2。
设置[通讯设置]与扩展串口匹配。

2 在[通用设置(R)]菜单中选择[扩展脚本设置(E)]。



注 释 • 这时将显示“扩展脚本将被启用，要继续吗？”，点击[是]。

3 将“INIT”注册为自定义函数。点击[函数]选项卡并点击自定义函数框的[创建]按钮。



4 输入 [INIT] 作为函数名，点击 [确定]。将显示如下画面。

5 用命令、语句和常量在执行表达式中创建脚本。

脚本表达式区 放大脚本表达式区域 输入地址

```
0001 [c:EXT_SIO_CTRL00]=1    //Send buffer clear
0002 [c:EXT_SIO_CTRL01]=1    //Receive buffer clear
0003 [c:EXT_SIO_CTRL02]=1    //Error buffer clear
0004
```

6 以同样的方式将“PINIT”注册为自定义函数。输入 [PINIT] 作为函数名并在执行表达式中创建如下脚本。

脚本表达式区 放大脚本表达式区域 输入地址

```
0001 Call Strset                  //call Printing Data function
0002 _strset(databuf0,"")         //clear databuf0
0003
0004 //Printing and Delimiter (Carriage Return / Line Feed)
0005
0006 _strset(databuf0,0x0d)       //Print out ,then go back to first place of the line
0007 _strcat(databuf1,databuf0)   //append databuf0 into the end of databuf1
0008 _strset(databuf0,"")         //clear databuf0
0009 _strset(databuf0,0x0a)       //go to next line
0010 _strcat(databuf1,databuf0)   //append databuf0  into the end of  databuf1
0011
0012 _strlen([t:0000],databuf1)   //store data length to temporary address
0013
0014 //Send data over serial port
0015
0016 IO_WRITE_EX([p:EXT_SIO],databuf1,[t:0000])   //Send databuf0, amount defined by temporary address value
0017
```

7 以同样的方式将“Strset”注册为自定义函数。输入 [Strset] 作为函数名并在执行表达式中创建如下脚本。

脚本表达式区 放大脚本表达式区域 输入地址

```
0001 //String example, add "Price:" and "$"
0002 _strset(databuf0, "")                          //Initialize databuf0
0003 _strset(databuf0, "Price:")                    //Store text "Price:" to databuf0
0004 _bin2decasc(databuf0,[w:[#MEMLINK]0500])       //Convert value to string and store in databuf1
0005 _strcat(databuf0, databuf1)                    //Add databuf1 to end of databuf0
0006 _strset(databuf1, "")                          //Initialize databuf1
0007 _strset(databuf1, "$")                         //Store text "$" to databuf1
0008 _strcat(databuf0, databuf1)                    //Add databuf1 to end of databuf0
0009                                                                                          1
0010 //Initialize temporary address
0011 [t:0001]=0
0012 [t:0002]=0
0013
0014 //Store to internal memory word units, consecutive characters into byte units(30 characters)
0015 loop()
0016 {
0017  [w:[#MEMLINK]2000]#[t:0002]=[w:[#MEMLINK]1000]#[t:0001]>>8      //Store top byte into bottom byte
0018  [w:[#MEMLINK]2001]#[t:0002]=[w:[#MEMLINK]1000]#[t:0001]& 0*FF   //Erase top byte and store in next address
0019  [t:0001]=[t:0001]+1                                            //Address offset + 1
0020  [t:0002]=[t:0002]+2                                            //Address offset + 2
0021  if([t:0001]==15)                                               //Store 2 words into 2 byte and repeat 15 times
0022  {                                                                                       2
0023   break
0024  }
0025  endif
0026 }
0027 endloop
0028 _ldcopy(databuf2, [w:[#MEMLINK]2000],30)        //Store internal memory 2000~2030 to data buffer as characters
0029
0030 //Add string "Item:"
0031 _strset(databuf1, "")                           //Initialize databuf1
0032 _strset(databuf1, "Item:")                      //Store string "Item:" into databuf1
0033 _strcat(databuf1, databuf2)                     //Add databuf1 to end of databuf0
0034                                                                                          3
0035 //Add Item and Price strings
0036 _strcat(databuf1, databuf0)                     //Add databuf0 to end of databuf1
```

8 以同样的方式将“Print”注册为自定义函数。输入 [Print] 作为函数名并在执行表达式中创建如下脚本。

脚本表达式区 放大脚本表达式区域 输入地址

```
0001 Call Street                        //Call string data function
0002 _strset(databuf0,"")              //Clear databuf1
0003
0004 //Text delimiter
0005
0006 _strset(detabuf0, 0*0d)           //Return to start of row
0007 _strset(databuf1, databuf0)       //Add databuf1 to end of databuf0
0008 _strset(databuf0, "")             //Clear databuf1
0009 _strset(databuf0, 0*0a)           //New line
0010 _strset(databuf1, databuf0)       //Add databuf1 to end of databuf0
0011
0012 _strset([t:0000], databuf1)       //Store data length to temporary address
0013
0014 //Send data over serial port
0015
0016 IO_WRITE_EX([p:EXT_SIO], databuf1, [t:0000] //Send databuf0, amount defined by temporary address value
0017
```

9 创建主脚本。在执行表达式中创建如下脚本，完成设置。

脚本表达式区 放大脚本表达式区域 输入地址

```
0001 //Receive 1 byte of printable data from printer
0002 if([r:EXT_SIO_RECV]==1)                                    //When received data is 1
0003 {
0004     _strset(databuf0,"")                                   //Initialize databuf0
0005     IO_READ_EX([p:EXT_SIO], databuf0, 1)                   //Read data into databuf0
0006 _dlcopy([w:[#MEMLINK]0100], databuf0, 0, 1)                //Store values from databuf0 to internal memory
0007 }
0008 endif
0009
0010 //Determine whether to print data
0011 if([b:[#MEMLINK]005000]==1 and [w:[#MEMLINK]0100]==0x31)   //Printer start switch is ON
0012 {                                                          //when 1 (ASCII)
0013    Call INIT                                               //call communication INIT function
0014    Call PINIT                                              //call printer INIT function
0015    Call Print                                              //send print data, call print function
0016    clear([b:[#MEMLINK]005000]                              //Printer start switch OFF
0017 }
0018 endif
0019
0020 if([b:[#MEMLINK]005000]==1 and [w:[#MEMLINK]0100]==0x30)   //Printer start switch is ON
0021 {                                                          //when 0 (ASCII)
0022    clear([b:[#MEMLINK]005000]                              //Printer start switch OFF
0023
0024 }
0025 endif
```

注 释

- 要将步骤 3 至 9 中创建的自定义函数放到主脚本中，请选择要放置的函数，然后点击 [函数] 选项卡上的 [调用]。将使用 “调用函数名” 放置函数。
- 当选择文本时，按 [Ctrl] 键 + [Shift] 键 +[右箭头] 键 /[左箭头] 键来选择整个文本块。
- 按 [Ctrl] 键 + [F4] 键来关闭当前选择的画面。
- 按 [Esc] 键来覆盖并保存脚本或删除脚本并退出。

21.6 参考其他脚本

21.6.1 简介

可以并排显示自定义函数和 D 脚本、全局 D 脚本、扩展脚本或另外一个用户定义函数。
可以在比较它们的同时编写函数，也可以同时编辑两者。

注 释

- 也可以同时编辑不同类型的自定义脚本。

21.6.2 操作步骤

注 释

- 更多详情，请参阅 “设置指南”。

☞ "21.9.1 [D 脚本] / [通用设置] - [全局 D 脚本设置] 设置指南 " (p21-52)

- 有关脚本命令的信息，请参阅以下内容。

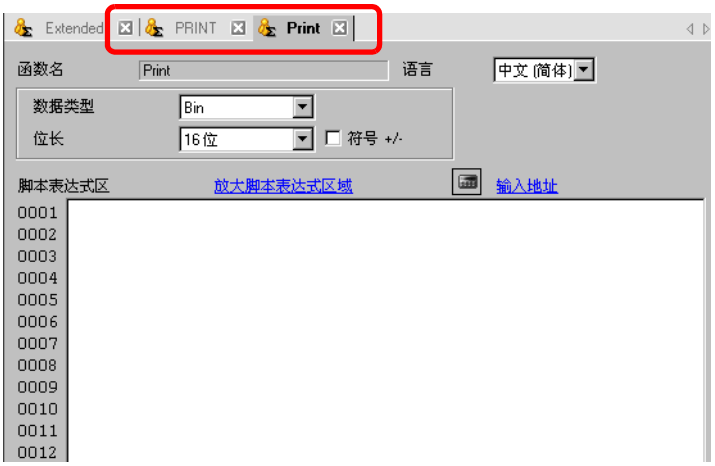
☞ "21.11 指令 / 条件表达式 " (p21-65)

- 有关脚本命令的信息，请参阅以下内容。

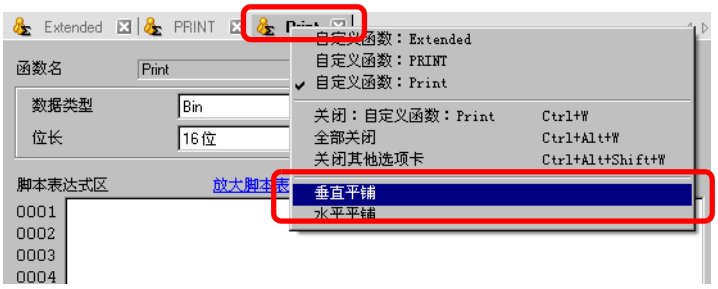
☞ "21.11 指令 / 条件表达式 " (p21-65)

在 [D 脚本] 对话框上，将画面水平或垂直地拆分为两个画面。
例如，在显示先前创建的函数 “PRINT” 的同时，创建 “Print”。

1 在 [D 脚本] 对话框上，打开想要同时显示的脚本和自定义函数。



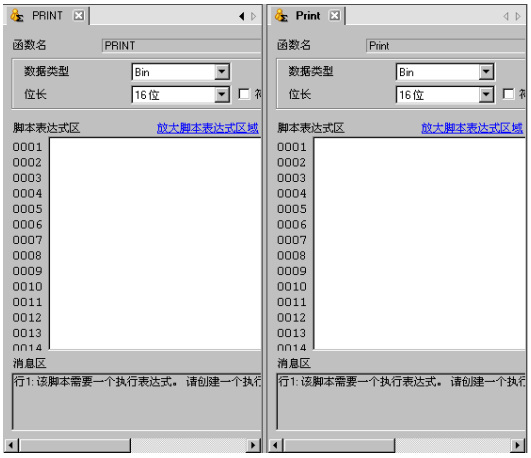
2 如果要水平平铺，右击准备将其显示在下侧的画面的选项卡，然后点击 [水平平铺]。



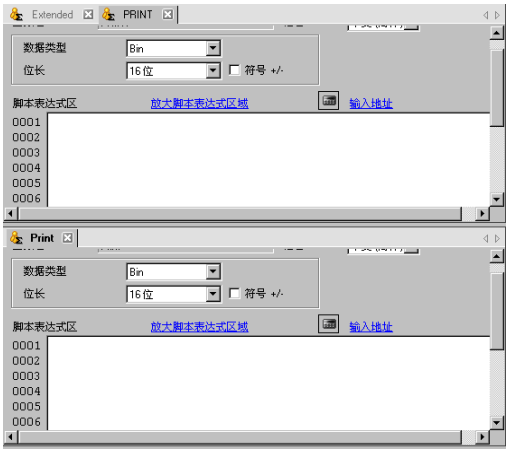
如果要垂直平铺，右击准备将其显示在右侧的画面的选项卡，然后点击 [垂直平铺]。

3 画面水平或垂直平铺显示。

水平平铺



垂直平铺



21.7 脚本创建步骤

21.7.1 D 脚本 / 全局 D 脚本创建步骤

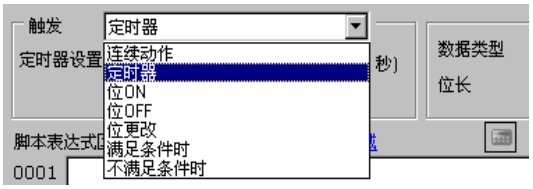
从 [部件 (P)] 菜单中选择 [D 脚本 (R)]。

点击 [创建]。要浏览现有脚本，选择 ID 编号并点击 [编辑] 或双击 ID 编号行。

在 [通用设置 (R)] 菜单中选择 [全局 D 脚本设置 (L)]。

点击 [创建]。要浏览现有脚本，选择 ID 编号并点击 [编辑] 或双击 ID 编号行。

设置导致脚本执行的触发条件。有关该函数的更多信息，请参阅 "21.8 触发条件设置 " (p21-45)。



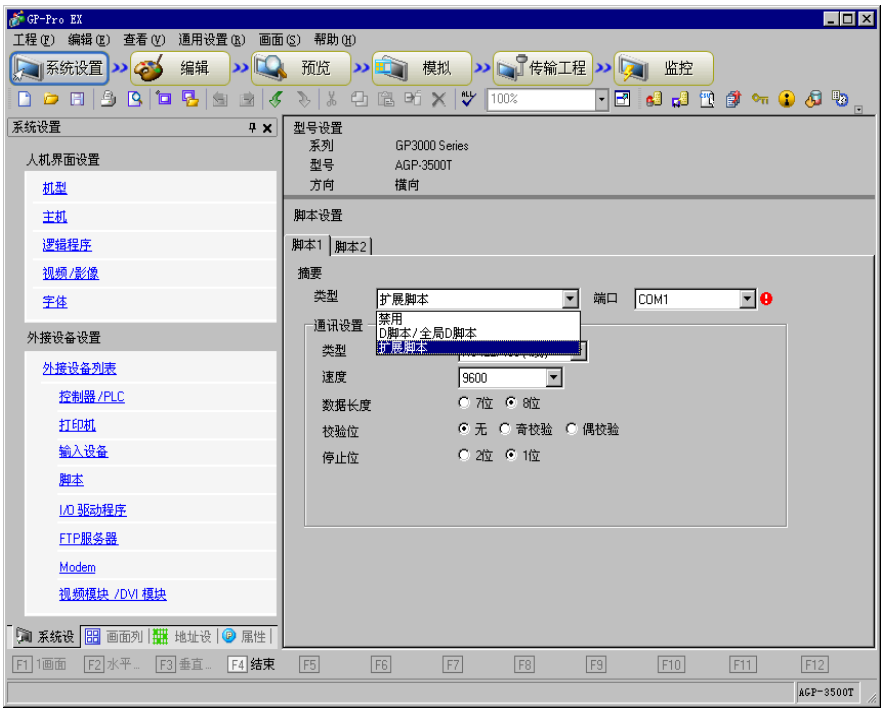
创建脚本 (执行表达式)。有关命令和函数的更多信息，请参阅 "21.11 指令 / 条件表达式 " (p21-65)。

注 释

- 组件托盘按 D 脚本的创建顺序显示已注册的 D 脚本。要更改组件托盘中的顺序，请更改注册项目的 ID 编号，然后从 [编辑] 菜单中选择 [自动对齐托盘]。双击组件托盘中的项目，显示它们的编辑对话框，可更改 ID 设置。

21.7.2 扩展脚本创建步骤

从 [工程 (F)] 菜单中选择 [系统设置 (C)] 。点击 [脚本]，然后将显示如下对话框。
当使用扩展脚本时，将 [类型] 设置为 [扩展脚本] 并选择适当的 [端口]。

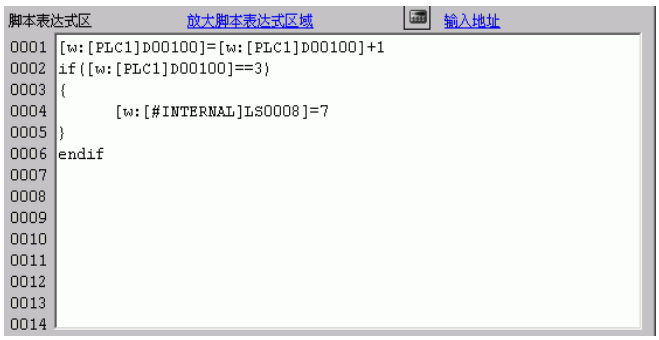


↓

在 [通用设置 (R)] 菜单中选择 [扩展脚本设置 (E)]。

↓

创建脚本 (执行表达式)。有关命令和函数的更多信息，请参阅 "21.11 指令 / 条件表达式 " (p21-65)。



21.7.3 自定义函数设置

将现有脚本注册为自定义函数，以便能在其他脚本中使用它。注册的函数可由 D 脚本、全局 D 脚本和扩展脚本使用。

■ 设置步骤

当创建新的自定义函数时
点击 [创建]。将显示自定义函数对话框。

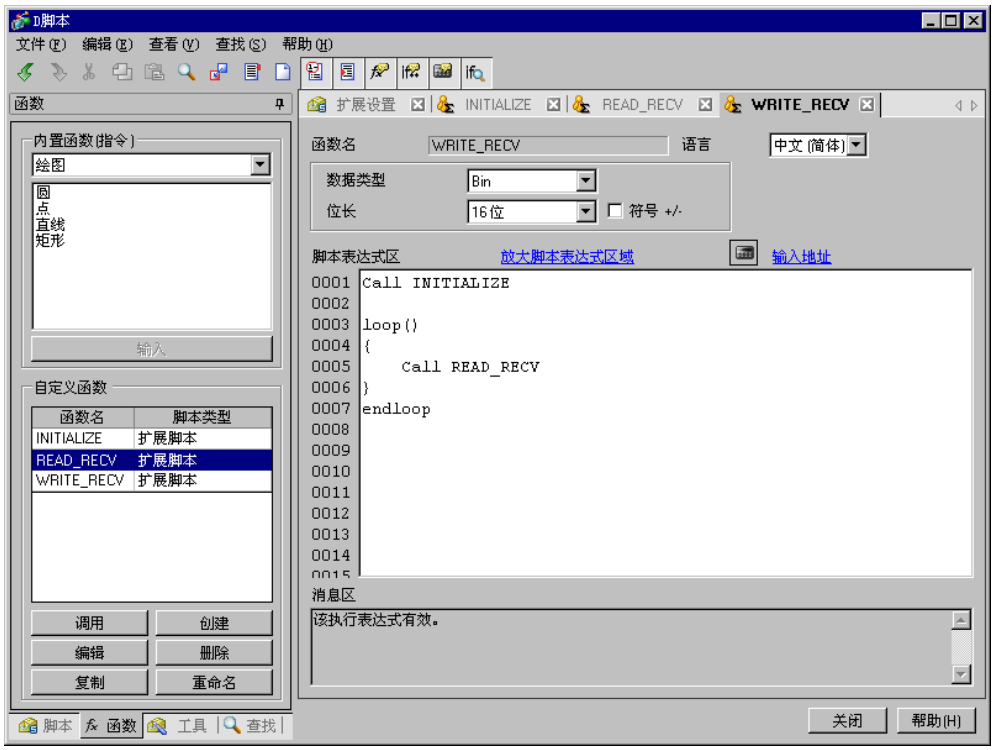
当编辑一个以前注册的自定义函数时
选择您想修改的自定义函数并点击 [编辑]。将显示自定义函数对话框。

输入函数名并在执行栏中创建脚本。点击 [确定]，保存自定义函数。

注 释

- 函数名有限制。更多信息，请参阅 "21.10.3 自定义函数限制 " (p21-62)。

选择要调用的自定义函数，点击 [调用]，会将 “调用函数名”放在执行栏中。



21.8 触发条件设置

已创建的脚本可以使用以下 7 种触发条件中的任意一种。

设置		描述
连续操作		脚本被定期触发。
定时器		脚本在经过指定的时间后被触发。
位	位 ON	当 GP 检测到指定位从 0 变为 1 时，脚本被触发。
	位 OFF	当 GP 检测到指定位从 1 降为 0 时，脚本被触发。
	位更改	当 GP 检测到指定位从 0 到 1 或从 1 到 0 的变化时，脚本被触发。
条件表达式	当条件成立时	当 GP 检测到指定表达式成立时，脚本被触发。
	当条件不成立时	当 GP 检测到指定表达式不成立时，脚本被触发。

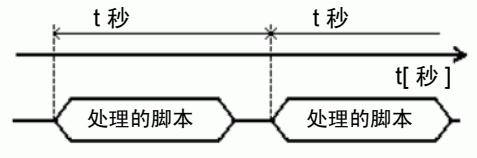
21.8.1 连续操作

每个显示扫描时间执行。

21.8.2 定时器

■ 定时器

每次经过指定的时间时，执行脚本一次。定时器持续时间可以在 1 到 32767 秒之间进行设置。



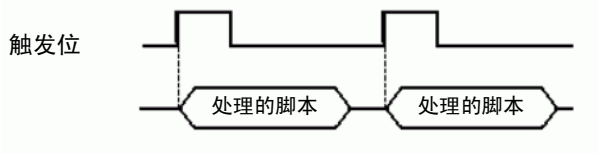
注 释

- 当设置定时器功能的时间时，时间值包括设置时间 + 显示扫描时间误差。此外，根据绘制画面项目或打印数据所需的时间，定时器功能可能会变慢。有关显示扫描时间的更多信息，请参阅 " ■ 触发位限制 " (p21-49)。
- 在使用 D 脚本时，切换画面将造成定时器功能从 0 开始重新计时。

21.8.3 位

■ 位 ON

当 GP 检测到指定位地址 (触发位) 从 0 变为 1 时，脚本被触发。

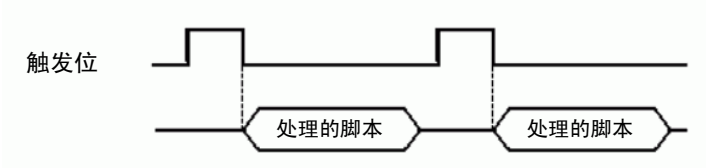


注 释

- 对于触发位的 ON/OFF 转换，请务必留出一个时间间隔，该间隔至少为通讯周期时间和显示扫描时间之中的较大者。有关该功能的更多信息，请参阅 " ■ 触发位限制 " (p21-49)。

■ 位 OFF

当 GP 检测到指定位地址 (触发位) 从 1 变为 0 时，脚本被触发。



注 释

- 对于触发位的 ON/OFF 转换，请务必留出一个时间间隔，该间隔至少为通讯周期时间和显示扫描时间之中的较大者。有关该功能的更多信息，请参阅 " ■ 触发位限制 " (p21-49)。

■ 位更改

当 GP 检测到指定位地址 (触发位) 从 0 到 1 或从 1 到 0 的变化时，脚本被触发。

注 释

- 对于触发位的 ON/OFF 转换，请务必留出一个时间间隔，该间隔至少为通讯周期时间和显示扫描时间之中的较大者。有关该功能的更多信息，请参阅 " ■ 触发位限制 " (p21-49)。

21.8.4 条件表达式

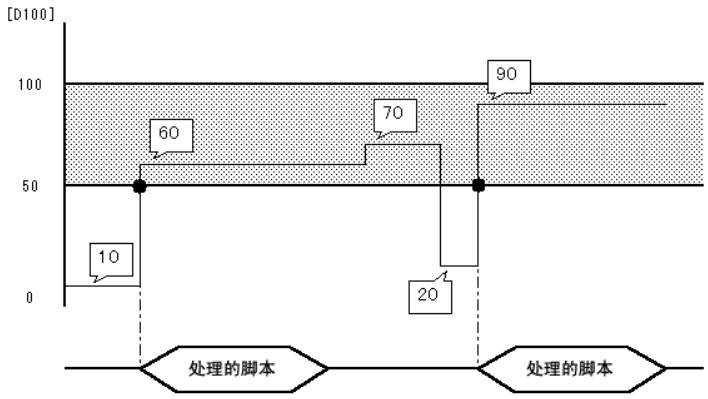
■ 当条件成立时

当 GP 确定触发条件成立时，脚本运行一次。

当触发条件设置为 100>[D100]>50 时，脚本以如下时序运行。

检测到 [假]--> [真]，脚本执行，将 70 赋值给 D100。

从 [真]-->[真] 时不执行脚本。



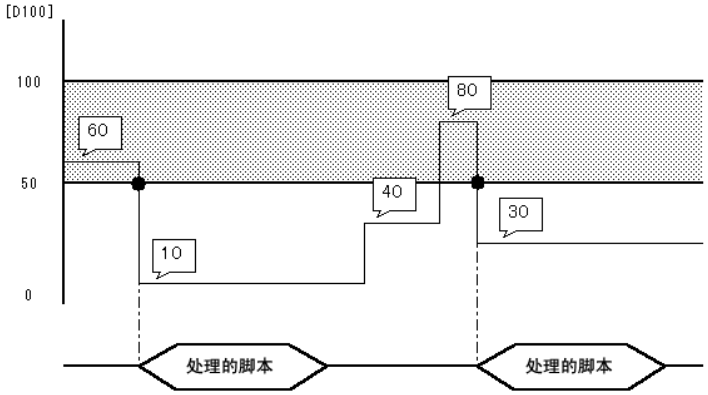
注 释

- 对于触发条件，需留出一个时间间隔，该间隔应大于通讯周期时间和显示扫描时间两者中的较大者。有关该功能的更多信息，请参阅 " ■ 触发位限制 " (p21-49)。

■ 当条件不成立时

当 GP 检测到触发程序中的指定表达式不成立时，脚本执行一次。
当触发条件设置为 100>[D100]>50 时，脚本以如下时序运行。

检测到 [假]--> [真]，脚本执行，将 20 赋值给 D100。
从 [真]-->[真] 时不执行脚本。



注 释

- 对于触发条件，需留出一个时间间隔，该间隔应大于通讯周期时间和显示扫描时间两者中的较大者。有关该功能的更多信息，请参阅 " ■ 触发位限制 " (p21-49)。

■ 触发位限制

- 请务必留出一个比通讯周期时间长的间隔，以便在外接设备中执行写入操作。当通过使用 GP 内部特殊继电器的扫描计数器频繁执行到外接设备上的写入操作时，可能导致通讯错误或系统错误。
- 如果用于 D 脚本触发条件的位被设置为 “触摸”且该位在 D 脚本处理过程中置 OFF，反复点按触摸区时可能无法检测到位 ON 的上升沿。此时 D 脚本会将以前读取的值和当前读取的值进行比较，以确定该触发条件是否为 “真”。但是，在一次扫描过程中，保存在触发操作过程中使用的位地址中的值将保持不变，即使该值在执行过程中被更改。仅在下一次扫描开始后才会读取新值。

通讯周期时间：　通讯周期时间是指从人机界面从控制器 /PLC 请求数据一直到人机界面收到数据的这一段时间。它以二进制数据形式保存在内部寄存器地址区 LS2037 中。单位为毫秒 (ms)。有 ± 10ms 的误差。

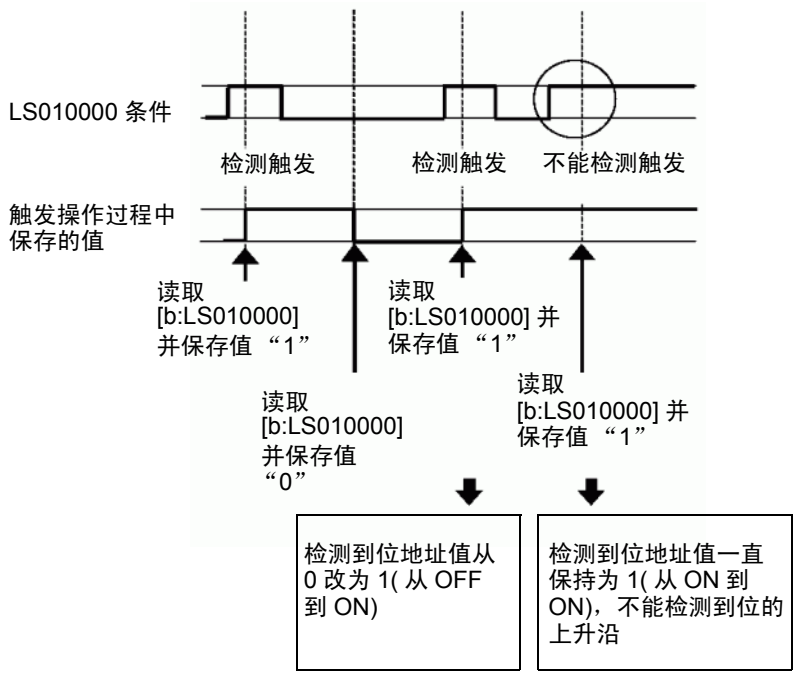
显示扫描时间：　显示扫描时间是处理一个画面所需的时间。它以二进制数据形式保存在内部寄存器地址区 LS2036 中。单位为毫秒 (ms)。有 ± 10ms 的误差。

例如，使用触摸来将触发位 (LS010000) 置 ON， D 脚本将该值置 OFF：

触发条件：位 ON [#INTERNAL] LS010000

执行表达式：clear ([b:[#INTERNAL]LS010000])

◆ D 脚本处理时序图



例如，如果未使用 D 脚本触摸时序而只是执行检测，处理如下。

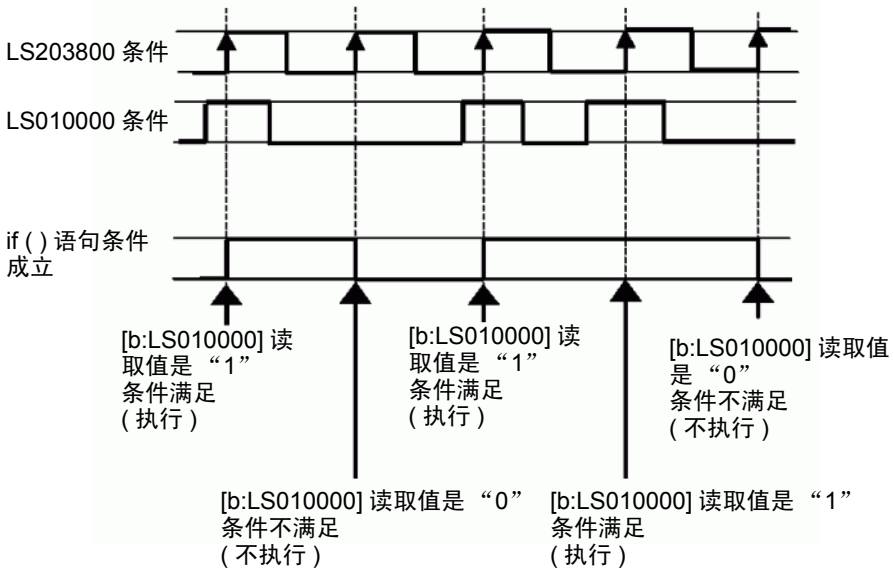
使用 if() 语句来检测触发：
使用 if 语句来决定触摸操作是否进行了置位。每次运行 if() 语句时，读取数值并进行比较检查。

触发条件：位 ON ([#INTERNAL]LS203800 *1)
执行表达式：if ([b:[#INTERNAL]LS010000]==1)
{
clear ([b:[#INTERNAL]LS010000])
:
:

*1 GP 内部计数器。每次当显示画面上设置的部件进行处理时计数器会增加一次。

当使用以前的 D 脚本时，即使连续触摸，脚本也只有在条件满足时才运行。如下面的时序图所示，每次显示扫描时读取一次数值并进行匹配检查，如果匹配，无论以前的值如何，脚本都会运行。

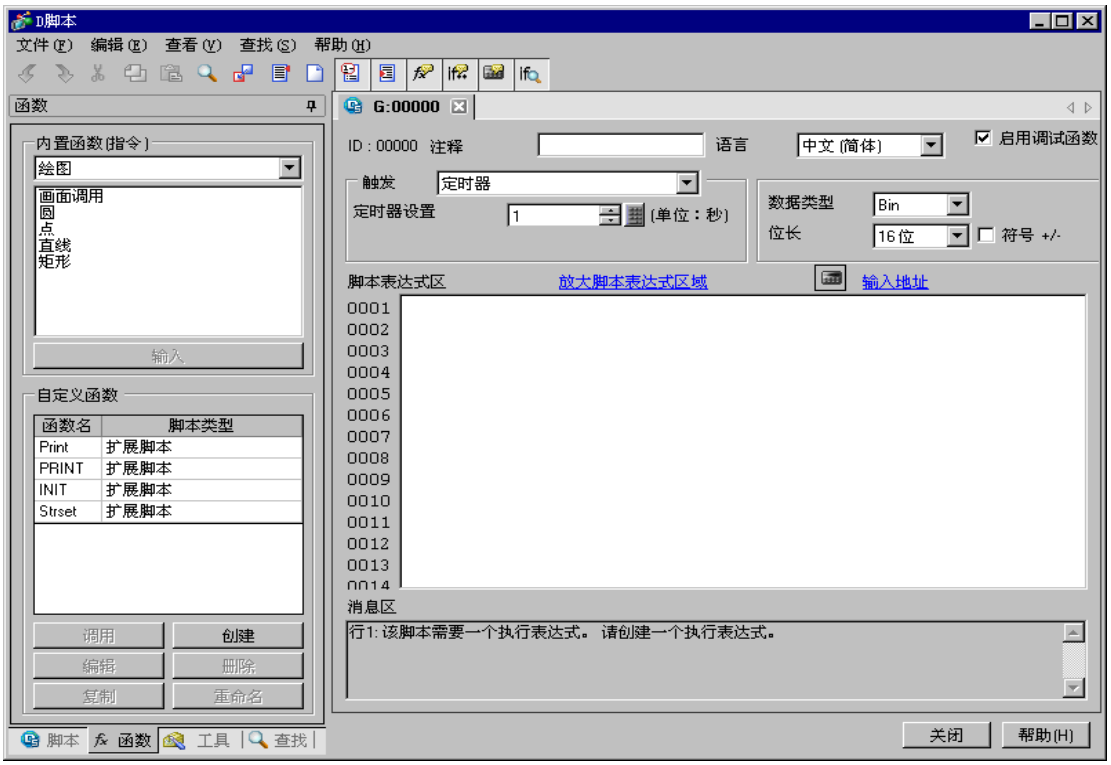
◆ D 脚本处理时序图

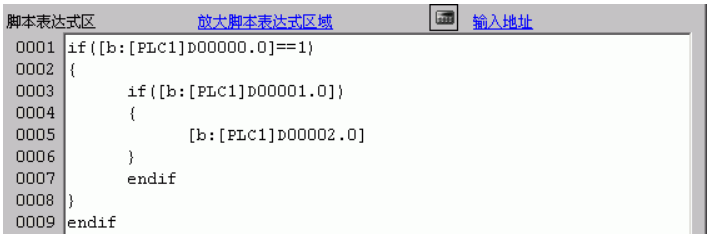

LS203800 条件
LS010000 条件
if ( ) 语句条件
成立
[b:LS010000] 读
取值是 “1”
条件满足
(执行)
[b:LS010000] 读
取值是 “1”
条件满足
(执行)
[b:LS010000] 读取值
是 “0”
条件不满足
(不执行)
[b:LS010000] 读取值是 “0”
条件不满足
(不执行)
[b:LS010000] 读取值是 “1”
条件满足
(执行)

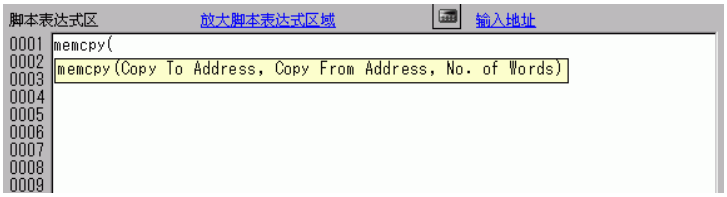
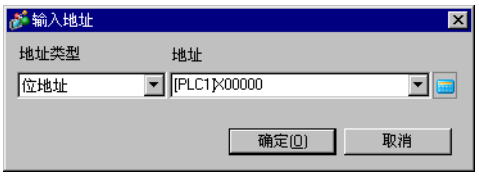
21.9 设置指南

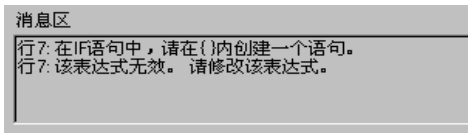
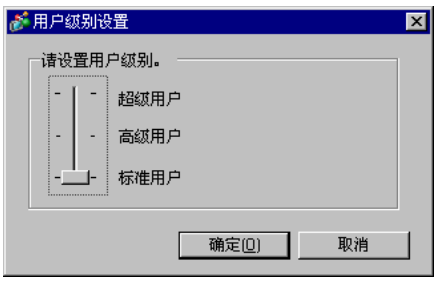
21.9.1 [D 脚本] / [通用设置] - [全局 D 脚本设置] 设置指南

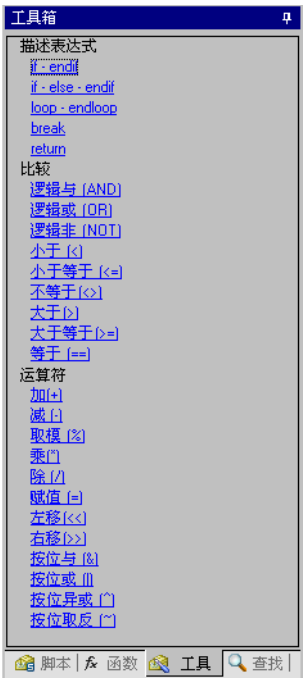
下面是通用 [全局 D 脚本] 对话框。您可以为 D 脚本指定的设置同对话框中的设置相同。对于 [通用设置] - [扩展脚本设置]，不指定 ID 和触发设置，但其他设置相同。



设置	描述
导出	这可以从文件菜单中选择。导出操作将已创建的脚本写入一个文本文件 (.txt)，然后可将该文本文件导入其他脚本。
导入 	这可以从文件菜单中选择。导入操作读入一个导出脚本 (文本文件)。
行号	在程序右侧显示行号。
自动缩进控制 	自动缩进语句如下。 

设置	描述
函数输入帮助	当如下所示输入函数和起始扩号 “(” 时，将显示出函数的格式。 
自动句法完成	当从键盘输入 “if” 或 “loop” 时，会自动放置剩余语句。
地址输入	创建脚本时，输入左方括号 ([) 可显示 [输入地址] 对话框。  从 [位地址]、 [字地址]、 [临时地址] 中选择地址类型。 • 位地址 可以指定控制器 /PLC 地址、 GP 内部寄存器和位变量。 • 字地址 可以指定控制器 /PLC 地址、 GP 内部寄存器和整型变量。 • 临时地址 该地址只能用于脚本。 请参阅如下内容获取内部寄存器的详细信息。 ☞ "A.1.2 使用 Direct Access 方式与控制器 /PLC 通讯 " (pA-3) ☞ "A.1.3 使用 Memory Link 方式与不支持的控制器 /PLC 通讯 " (pA-5) **重 要** • 在脚本中，请不要设置任何以 “0” 开始的密码等。所有以 “0” 开始的数值均被作为八进制 (base-8) 数据处理。 • 如何描述不同的输入数据格式 例如： • DEC (Base-10) : 非零开始值 例如， 100 • HEX(Base-16) : 以 0x 开始的值 例如， 0x100 • OCT (Base-8) : 以 0x 开始的值 例如， 0100 • 示例：使用运算符 AND 在不同的数据格式间 (Hex 和 BCD) 执行运算 仅 Hex 0x270F & 0xFF00 结果：0x2700 BCD 和 Hex 9999 9999 & 0xFF00 结果：0x9900

设置	描述
自动句法分析	脚本创建过程中检查句法。检查结果将显示在窗口的底部。 
ID	脚本按 ID 编号进行管理。 当创建具有不同触发条件的多个脚本时，设置一个 0 至 65535 之间的值。
注释	输入脚本注释。
语言	从下拉列表中选择一种语言：[日语]、 [ASCII]、 [中文 (简体)]、[中文 (繁体)]、 [韩语]。
启用调试函数	设置是否启用调试函数。如果脚本主体中有 _debug 函数，将执行 _debug 函数。 有关该功能的更多信息，请参阅 " ■ 调试函数 " (p21-129)。
锁定密码	选择是否启用锁定密码功能。 (1)勾选该复选框，显示 [用户级别设置] 画面。 (2)从 [标准用户]、 [高级用户] 和 [超级用户] 中选择用户级别，然后点击 [确定]。  (3)输入 [密码]，点击 [确定]。如果勾选 [记住密码] 复选框，会保存密码且不再显示密码解锁对话框。 
触发	设置导致脚本执行的触发条件。有关该功能的更多信息，请参阅 "21.8 触发条件设置 " (p21-45)。 扩展脚本没有触发条件设置。
数据类型	将脚本的数据格式设置为 Bin 或 BCD。 对扩展脚本来说，固定为 Bin 格式。
位长	将脚本的数据长度设置为 16 位或 32 位。
符号 +/-	当您想插入负数时选择该项。 只有当数据类型是 Bin 时才可以设置它。
执行表达式	脚本内容。

设置	描述
内置函数 (指令)	从工具栏中选择命令和函数，可轻松地将它们添加到脚本中。 有关可用命令和函数的更多信息，请参阅 ☞"21.11 指令 / 条件表达式 " (p21-65) 内置函数  从 [内置函数 (指令)] 中选择一个类别。在底部会显示相关函数。 选择函数并点击 [输入]。将显示相应的设置对话框。
自定义函数	将一个脚本注册为自定义函数，它可以由其他脚本使用。  注 释 • 有关自定义函数的详细信息，请参阅 "21.9.2 [自定义函数] 设置指南 " (p21-56)。
工具箱	作为一个快捷键，从工具箱中选择在脚本中使用的命令。 此外，还可以选择在脚本中查找和定位文本的命令。 有关可用命令的更多信息，请参阅 "21.11 指令 / 条件表达式 " (p21-65)。 

21.9.2 [自定义函数] 设置指南

设置	描述
函数名称	显示自定义函数的名称。
脚本类型	显示脚本类型。使用下拉菜单，可以在 [D 脚本] 或 [扩展脚本] 之间切换。
调用	调用一个已创建函数。选择要调用的函数，点击 [调用]，会将 “调用函数名” 放在表达式区。
创建	创建一个新函数。点击 [创建]。将显示 [函数名] 对话框。
编辑	编辑现有函数。选择要编辑的函数，点击 [编辑]。将显示 [D 脚本函数] 对话框。
删除	删除现有函数。选择要删除的函数，点击 [删除]。
多重复制	复制现有函数。选择要复制的函数，点击 [复制] 来显示对话框，创建函数副本。
重命名	更改现有函数的名称。点击 [重命名]。将显示重命名函数对话框。

21.10 限制

21.10.1 D 脚本 / 全局 D 脚本限制

- 在 D 脚本编程中，三个地址与一个部件占用的内存一样多。 D 脚本的最大可用地址数是 255[*1]。尽可能少地使用地址，因为使用的寄存器越多，响应速度越慢。
- D 脚本不能对浮点型数值 (浮点型变量或实型变量) 执行计算。也不能对结构变量执行计算。但是，可以对结构变量的单个元素执行计算。
- D 脚本的大小影响显示扫描时间。需要注意的是，使用大量地址可能会极大地削弱程序性能。
- 执行写入控制器 /PLC 地址时，请勿为脚本的触发条件指定 [连续操作]。否则，由于通讯处理不能跟上大量的写入指令，会显示错误消息。要启用 [连续操作]，请使用 GP 内部寄存器或临时地址。
- 当在函数内调用函数时，嵌套层的最大数量是 9。请勿创建 9 个以上的嵌套层。
- 最多可以创建 9 个嵌套层。
- 最多可以创建 254 个函数。

◆ 根据为触发条件指定的寄存器，画面切换后由触发激活的 D 脚本操作如下：

触发条件	除 [#MEMLINK] 外的任何外接控制器				[#MEMLINK]			
当前值或条件	位 “0”	位 “1”	条件不满足	条件满足	位 “0”	位 “1”	条件不满足	条件满足
位的上升沿	X	O	-	-	X	X	-	-
位的下降沿	O	X	-	-	X	X	-	-
位更改	O	O	-	-	X	X	-	-
定时器设置	X	X	X	X	X	X	X	X
检测真	-	-	X	O	-	-	X	O
检测假	-	-	O	X	-	-	O	X

O: 在画面切换或接通电源后立即执行操作。

X : 在画面切换或接通电源后不立即执行操作。

- 当定时器运行时，定时器在画面切换后立即开始计时。
- 当使用全局 D 脚本时，仅当 GP 上电时执行上述操作。但是，当 GP 画面切换时，不执行上述操作，并继续监视触发条件。
- 当全局 D 脚本包含定时器时，定时器在 GP 上电后立即开始计时。

注 释

- 不要使用触摸屏键来设置触发位或操作程序中的开始位。因为触摸输入的时机可能不正确，从而导致不正确的位输入。

*1 触发表达式和脚本程序中使用的寄存器总数。

◆ **如果在执行 D 脚本命令的同时将一个值分配给一个用于切换画面的地址时，将在所有 D 脚本都处理完成后才处理画面切换操作。**

例如：

```
ID              00000
数据类型        Bin          数据长度   16 位      符号 +/-    无
触发            上升位 ([b:M0000])
[w:[PLC1]D0100]=0              // (1)
[w:[#INTERNAL]LS0008]=30       // (2) 切换至基本画面号 30
[w:[PLC1]D0101]=1              // (3)
[w:[PLC1]D0102]=2              // (4)
```

当执行了上述 D 脚本时，画面切换处理在处理完 (3) 和 (4) 后进行。

◆ **如果 D 脚本中使用的数据是使用 GP 触摸操作创建的，确保数据写入操作在运行该 D 脚本前已完成。**

◆ **全局 D 脚本的特定限制**

- 当 GP 上电时，执行上页表中显示的操作。当画面切换时，上表不适用，会继续监视触发条件。
- 画面切换或其他 GP 操作过程中全局 D 脚本操作暂停。
- GP 上电后，全局 D 脚本操作直到初始画面的所有数据读取完毕后才执行。在初始画面切换后，全局 D 脚本操作可以在数据读取完成前执行。
- 全局 D 脚本内寄存器的最大数量是 255[*1]。当超出这一数量时， D 脚本不起作用。由于这些寄存器在无论画面为何均读取数据，因此请务必在 D 脚本中使用尽可能少的寄存器。否则可能影响操作性能。
- 最多可使用 32 个全局 D 脚本。当前使用的函数也算作一个全局 D 脚本。当全局 D 脚本数达到 32 时，将忽略所有后续的全局 D 脚本。

*1　触发表达式和脚本程序中使用的寄存器总数。

◆ 串口端口操作的限制

- 发送 / 接收函数中指定的地址不算作 D 脚本地址。
- 控制变量是只写变量，而状态和已接收数据是只读变量。读取控制变量或将数据写入状态变量会导致操作失败。
- 请为发送和接收操作创建单独的 D 脚本 (或函数)。有关数据传输流程图的更多信息，请参阅
  ☞" ■ 流程图 " (p21-24)
- LS 寄存器 (LS20 至 LS2031 和 LS2096 至 LS8191) 中的用户区可以为发送 / 接收函数保存数据。
- 在 [系统设置] 中的 [脚本] 设置页面，若未将 [类型] 设置为 [D 脚本 / 全局 D 脚本]，地址 LS2032 中的第 13 个位在 [D 脚本 / 全局 D 脚本] 运行 [串口操作] 的标签设置功能时 (发送、接收、控制、读取状态和接收数据大小) 置 ON。有关特殊继电器的更多信息：
  ☞"A.1.4.3 特殊继电器 " (pA-19)
- 当使用发送 / 接收函数时，将 D 脚本的位长设置为 16 位。需要注意的是，如将位长设置为 32 位，操作就会失败。
- 发送缓冲器的大小是 2048 字节，而接收缓冲器的大小是 8192 字节。在至少 80% 的接收缓冲器充满接收数据后， ER 信号 (输出)RS 信号 (输出) 置 OFF。

◆ BCD 格式运算的限制

如果运算过程中发现了一个不能转换成 BCD 格式的值，程序会停止运行。
这些值包括十六进制格式中的 A 至 F。
请勿使用这样的值。如果程序由于非 BCD 值而停止， GP 通用继电器信息 (LS2032) 中的位 7 置 ON。该位直到 GP 关机或进入离线状态时才置 OFF。
例如：

[w:[PLC1]D0200]=([w:[PLC1]D0300]<<2)+80
如果 D300 是 3，左移两位的结果是 0x000C，不能将其转换成 BCD 格式，因此中断程序运行。

[w:[PLC1]D0200]=[w:[PLC1]D0300]<<2 如果 D300 是 3，左移两位会得到 0x000C。
与上面的例子不同， 0x000C 是将保存在内存中的运算结果，不会造成程序停止。

◆ 零运算的限制

如果在除法 (/) 和取模 (%) 运算中除以零，程序将停止。请勿除以 0。
如果程序由于上述错误而停止， GP 通用继电器 (LS2032) 中的位 8 置 ON。该位直到 GP 关机或进入离线状态时才置 OFF。

◆ 赋值运算中的延迟注意事项

在赋值运算中使用寄存器地址可能造成写延迟，因为 GP 必须从外接控制器中读取地址数据。考虑如下的问题：
例如：

[w:[PLC1]D0200]=([w:[PLC1]D0300]+1 ...
[w:[PLC1]D0201]=([w:[PLC1]D0200]+1 ...

语句 (1) 将 (D0300+1) 赋值给 D0200。但是，在语句 (2) 中，由于与控制器 /PLC 的通讯耗时较长，语句 (1) 的结果尚未赋值给 D0200。在这种情况下 , 编程时应先将语句 (1) 的结果保存在 LS 区，如下所示。

[w:[#INTERNAL]LS0100]=[w:[PLC1]D0300]+1
[w:[PLC1]D0200] = [w:[#INTERNAL]LS0100]
[w:[PLC1]D0201]=[w:[#INTERNAL]LS0100]+1

◆ 处理负数时的注意事项

对于那些在其不接受负数的参数中输入了负数的函数来说，*1 输入的数将当作不带符号的数进行运算*2。

*1 例如， _CF_read() 参数的 “字节数”不能接受负数，因为它是被读取的数据大小。

*2 例如， -1 被当作 16 位的 65535 和 32 位的 4294967295 进行处理。

21.10.2 扩展脚本限制

- 对寄存器地址来说，只能使用 LS 区和 USR 区 (扩展用户区)。
- D 脚本和全局 D 脚本的临时地址的管理与扩展脚本临时地址的管理是分开的。因此，对 D 脚本和全局 D 脚本的临时地址的更改不会反映在扩展脚本的临时地址中。
- 可以调用由 D 脚本 / 全局 D 脚本创建的自定义函数，但是如果在该函数内访问内部寄存器以外的寄存器地址，它可能不会正常发挥作用。此外，在传输时 (为 GP 创建数据期间)，单独为 D 脚本、全局 D 脚本和扩展脚本创建自定义函数。
- 当从函数中调用函数时，最大的嵌套层数是 9 层。
- 最多可以调用 254 个函数。(用 “Call” 最多可调用 254 个函数。)
- 扩展脚本不影响标签计数。
- 如果由 D 脚本或全局 D 脚本调用仅由扩展脚本支持的函数 (例如字符串运算)，函数将不能发挥作用。
- 可用数据格式是 Bin。禁用 BCD 数据格式。
- 发送缓冲器的大小是 2048 字节，而接收缓冲器的大小是 8192 字节。在至少 80% 的接收缓冲器充满接收数据后， CTS 线置 OFF。
- 不能同时选择 D 脚本 / 全局 D 脚本和扩展脚本。注意下表中列出的组合。

扩展串口设置	D 脚本 / 全局 D 脚本 扩展脚本的扩展串口函数	扩展脚本的扩展串口函数
D 脚本 / 全局 D 脚本	O：可能的操作	X：将不操作
扩展脚本	X：将不操作	O：可能的操作

- 字符串设置的符号用法

当用 “_strset()” 和其他函数使用字符串时，需用双引号 (“”) 将字符串括起来。要显示字符串内的双引号，添加 “\” 符号并表示为 [\"]。不能表示单 “\” 符号。必要时，可使用字符代码格式设置 (_strset (databuf0, 92)。

例如：

"ABC\"DEF" ABC"DEF
"ABC\DEF" ABC\DEF
"ABC\\"DEF" ABC\"DEF
"ABC\\DEF" ABC\\DEF

- 对于那些在其不接受负数的参数中输入了负数的函数来说，*1 输入的数将当作不带符号的数进行运算 *2。

*1 例如， _CF_read() 参数的 “字节数” 不能接受负数，因为它是被读取的数据大小。

*2 例如， -1 被当作 16 位的 65535 和 32 位的 4294967295 进行处理。

◆ 专用扩展串口缓冲器的大小 (databuf0, databuf1, databuf2, and databuf3)

缓冲器	缓冲器名称	大小
数据缓冲器 0	databuf0	1 KB
数据缓冲器 1	databuf1	1 KB
数据缓冲器 2	databuf2	1 KB
数据缓冲器 3	databuf3	1 KB

21.10.3 自定义函数限制

- 可以使用的命令部分根据每个脚本而有不同。使用命令时，请参阅 "21.11 指令 / 条件表达式 " (p21-65)。
- 对于函数名，您可以使用任何英文字母或下划线字符 “_”。(但是，函数名必须以字母数字字符打头。)
- 请勿使用以下字符作为函数名。

and	b_call	Bcall	_bin2hexasc	break	Call
_CF_delete	_CF_dir	_CF_read	_CF_read_csv	_CF_rename	_CF_write
_USB_delete	_USB_dir	_USB_read	_USB_read_csv	_USB_rename	_USB_write
clear	databuf0	databuf1	databuf2	databuf3	_decasc2bin
_dlcopy	dsp_arc	dsp_circle	dsp_dot	dsp_line	dsp_rectangle
else	endif	fall	_hexasc2bin	if	IO_READ
IO_READ_EX	IO_READ_WAIT	IO_WRITE	IO_WRITE_EX	loop	_memcmp
memcpy	_memcpy_EX	memring	_memsearch	memset	_memset_EX
_memshift	not	or	return	rise	rise_expr
set	_strcat	_strlen	_strmid	_strset	timer
toggle	_wait				

21.10.4 运算结果注释

■ 溢出位

运算产生的溢出位采用四舍五入法处理。
对无符号 16 位数据执行运算时：

- 65535 + 1 = 0 (产生溢出位)
- (65534 * 2) / 2 = 32766(产生溢出位)
- (65534 / 2) * 2 = 65534 (不产生溢出位)

■ 余数处理的不同

余数处理的结果取决于是否在左侧和右侧添加了符号。

- -9 % 5 = -4
- 9 % -5 = 4

■ 被截取的小数位数

除法得到的小数值将被省去。

- 10 / 3 * 3 = 9
- 10 * 3 / 3 = 10

■ BCD 数据运算注意事项

产生溢出位的 BCD 数据运算不能得到正确的结果。

21.10.5 错误

当脚本配置错误时会显示如下错误消息。错误将显示在 GP 画面的底部。
在 LS91XX 地址中写入错误代码。在错误代码区中写入的数字是下表中 RAAA 后面的数字部分。(例如，当发生错误 RAAA130 时，将写入 ‘130’。)

脚本错误代码列表

D 脚本 (错误地址 =LS9120)	全局 D 脚本 (错误地址 =LS9110)	扩展脚本 (错误地址 =LS9100)
	RAAA130	RAAA140
未使用	全局 D 脚本错误。(全局 D 脚本的总数超过了最大数量 32。)	扩展 D 脚本错误 (函数总数超过了最大限值 255。)
	RAAA131	
未使用	全局 D 脚本错误。(寄存器总数超过了最大值 255*1。)	未使用
RAAA120	RAAA132	RAAA141
D 脚本错误 (指定的函数不存在或函数有错误。)	全局 D 脚本错误 (指定的函数不存在或函数有错误。)	扩展 D 脚本错误 (指定的函数不存在或函数有错误。)
RAAA121	RAAA133	RAAA142
D 脚本错误 (这些函数的嵌套达到 10 层以上。)	全局 D 脚本错误 (这些函数的嵌套达到 10 层以上。)	扩展 D 脚本错误 (这些函数的嵌套达到 10 层以上。)
RAAA122	RAAA134	RAAA143
D 脚本错误 (存在此版本不支持的表达式。)	全局 D 脚本错误 (存在此版本不支持的表达式。)	扩展 D 脚本错误 (存在此版本不支持的表达式。)
RAAA123	RAAA135	RAAA144
D 脚本错误 (在没有设置控制器 /PLC 的情况下使用串口运算函数。)	全局 D 脚本错误 (在没有设置控制器 /PLC 的情况下使用串口运算函数。)	扩展 D 脚本错误 (在没有设置控制器 /PLC 的情况下使用串口运算函数。)
RAAA124	RAAA136	RAAA145
D 脚本有错误。	全局 D 脚本有错误。	扩展 D 脚本有错误。

*1 触发表达式和脚本程序中使用的寄存器总数。

21.11 指令 / 条件表达式

■ 函数

项目	指令 / 函数	D 脚本 / 全局 D 脚本	扩展脚本
数据类型	Bin, BCD	O	仅 Bin
位长	16 位， 32 位	O	O
符号 +/-	启用 / 禁用	O	O
触发	定时器设置	O	X
	位的上升沿	O	X
	位的下降沿	O	X
	切换位	O	X
	表达式成立	O	X
	表达式为假	O	X
绘图	加载画面	O	X
	点	O	O
	直线	O	O
	圆形	O	O
	矩形	O	O
运算符	加 (+)	O	O
	减 (-)	O	O
	取模 (%)	O	O
	乘 (*)	O	O
	除 (/)	O	O
	赋值 (=)	O	O
比较	逻辑与	O	O
	逻辑或	O	O
	非 (NOT)	O	O
	小于 (<)	O	O
	小于等于 (<=)	O	O
	不等于 (<>)	O	O
	小于 (>)	O	O
	大于等于 (>=)	O	O
	等于 (==)	O	O

项目	指令 / 函数	D 脚本 / 全局 D 脚本	扩展脚本
存储器操作	复制存储器：memcpy ( )	O	O
	初始化存储器：memset()	O	O
	复制存储器 (变量指定) _memcpy_EX ( )	O	O
	初始化存储器 (变量指定) _memset_EX ( )	O	O
	偏移地址	O	O
	移位存储器	O	O
	存储器循环	O	O
	搜索存储器	O	O
	比较存储器	O	O
位操作	左移 (<<)	O	O
	右移 (>>)	O	O
	按位与 (&)	O	O
	按位或 (|)	O	O
	按位异或 (^)	O	O
	求反	O	O
	置位：set ( )	O	O
	复位：clear()	O	O
	反转：toggle ( )	O	O
条件表达式	if ( )	O	O
	if ( ) else	O	O
	loop ( ), break	O	O
	loop ( ) 无限循环	X	O
地址	位地址	O	内部寄存器
	字地址	O	内部寄存器
	临时工作地址	O	O*1
常量	Dec, Hex, Oct	O	O

项目	指令 / 函数	D 脚本 / 全局 D 脚本	扩展脚本
串口函数	接收： IO_READ ([p:SIO])	O	O
	发送：IO_WRITE ([p:SIO])	O	O
	扩展接收：_IO_READ_EX ( )	X	O
	扩展发送 _IO_WRITE_EX ( )	X	O
	待机接收功能 _IO_READ_WAIT ( )	X	O
	控制 [c:EXT_SIO_CTRL]	O	O
	状态 [s:EXT_SIO_STAT]	O	O
	已接收数据大小 [r:EXT_SIO_RCV]	O	O
	暂停：_wait ( )	X	O
文本操作	文本	X	O
	数据缓冲器 databuf0, databuf1, databuf2, databuf3	X	O
	写入字符串 _strset ( )	X	O
	从数据缓冲器到内部寄存器 _dlcopy ( )	X	O
	从内部寄存器复制到数据缓冲器：_ldcopy ( )	X	O
	十六进制文本至整数转换 _hexasc2bin ( )	X	O
	10 进制文本至整数转换 _decasc2bin ( )	X	O
	十六进制数字至字符串转换 _bin2hexasc ( )	X	O
	十进制数字至字符串转换 _bin2decasc ( )	X	O
	字符串长度 _strlen ( )	X	O
	字符串连接 _strcat ( )	X	O
	复制部分字符串 _strmid ( )	X	O
	状态 [e:STR_ERR_STAT]	X	O

项目	指令 / 函数	D 脚本 / 全局 D 脚本	扩展脚本
函数	Call	O	O
	return	X	O
CF 卡文件操作	读 CSV 文件	O	O
	输出文件列表 _CF_dir ( )	O	O
	读取文件 _CF_read ( )	O	O
	读 CSV 文件 CF_read_csv ( )	O	O
	写入文件 _CF_write ( )	O	O
	删除文件 _CF_delete ( )	O	O
	文件名更改 _CF_rename ( )	O	O
USB 文件操作	USB 读取文件	O	O
	输出文件列表 _USB_dir ( )	O	O
	读取文件 _USB_read ( )	O	O
	读 CSV 文件 USB_read_csv ( )	O	O
	写入文件 _USB_write ( )	O	O
	删除文件 _USB_delete ( )	O	O
	文件名更改 _USB_rename ( )	O	O
打印机操作	输出 COM 端口： IO_WRITE ([p:PRN])	O	O
调试	_debug ( )	O	O

*1 临时地址独立于 D 脚本和全局 D 脚本单独存在。

21.11.1 位操作

位操作	函数摘要
	位设置 ☞" ■ 位设置 " (p21-69) 从 0->1 更改指定位地址。
	清除位 ☞" ■ 清除位 " (p21-69) 从 1->0 更改指定位地址。
	位切换 ☞" ■ 位切换 " (p21-69) 从 1-> 0 或从 0-> 1 更改指定位地址。

■ 位设置

项目	描述
摘要	从 0->1 更改指定位地址。
格式	set ( )

表达式示例：

set ([b:[#INTERNAL]LS010000])

在上例中， LS0100 的第 00 位从 0->1。

■ 清除位

项目	描述
摘要	从 1->0 更改指定位地址。
格式	clear()

表达式示例：

clear ([b:[#INTERNAL]LS010000])

在上例中， LS0100 的第 00 位从 1->0。

■ 位切换

项目	描述
摘要	从 1-> 0 或从 0-> 1 更改指定位地址。
格式	toggle ( )

表达式示例：

toggle([b:[#INTERNAL]LS010000])

在上例中， LS100 的第 00 位从 1->0 或从 0->1 改变。

21.11.2 绘图

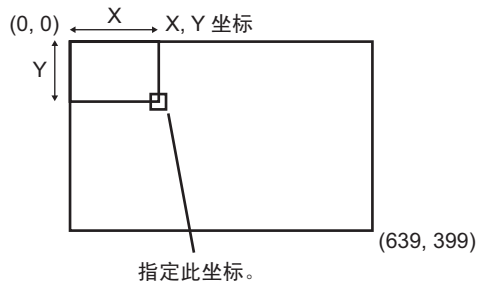
绘图	函数摘要
	调用画面 ☞" ■ 调用画面 " (p21-70) 调用指定画面号的画面 (基本画面)。 不能在扩展脚本中使用它。
	圆形 ☞" ■ 直线 " (p21-72) 绘制指定圆。
	点 ☞" ■ 点阵 " (p21-72) 绘制指定点。
	直线 ☞" ■ 直线 " (p21-72) 绘制指定直线。
	矩形 ☞" ■ 矩形 " (p21-73) 绘制指定矩形。

■ 调用画面

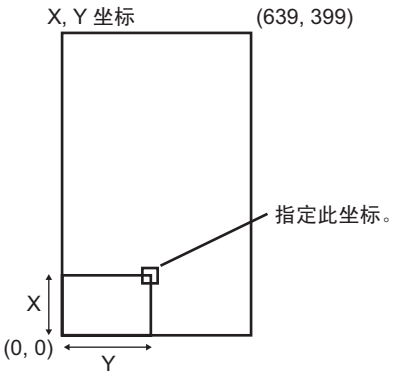
项目	描述
摘要	该功能调用已注册的库项目。在指定的 X、 Y 坐标上调用指定画面 (基本画面)。 不能在扩展脚本中使用它。
格式	b_call (画面号、 X 坐标、 Y 坐标)  注 释 • 用 X 坐标和 Y 坐标设置所调用画面的中心坐标。

坐标位置

横向型



纵向型



■ 圆形

项目	描述
摘要	在指定点上画一个圆。当勾选了 [图案] 复选框时，会绘制一个实心圆。选择并输入直线类型 (或在选择图案时选择实心图案)、颜色属性、中心坐标和半径值。可以间接设置中心坐标和半径。
格式	dsp_circle (X 坐标、 Y 坐标、半径、显示颜色闪烁 + 显示颜色、背景色闪烁 + 背景色、线型)  注 释 • 当同时设置了黑色和闪烁时，背景色会变成透明的。

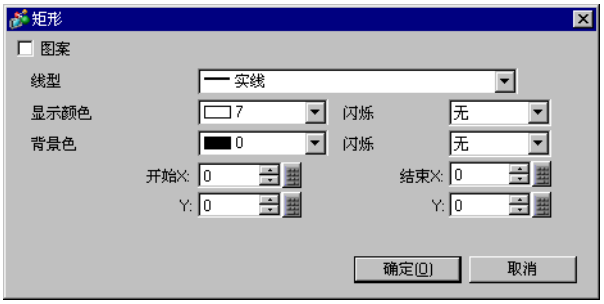
■ 点阵

项目	描述
摘要	在指定点上画一个点。设置 X、 Y 坐标及显示颜色。
格式	dsp_dot (X 坐标、 Y 坐标、闪烁 + 显示颜色)  注 释 • 当同时设置了黑色和闪烁时，背景色会变成透明的。

■ 直线

项目	描述
摘要	在指定点上画一条直线。设置直线类型、颜色属性及起始和结束坐标。
格式	dsp_line (起始点 X 坐标、起始点 Y 坐标、结束点 X 坐标、结束点 Y 坐标、显示颜色闪烁 + 显示颜色、背景色闪烁 + 背景色、线型和箭头)  注 释 • 当同时设置了黑色和闪烁时，背景色会变成透明的。

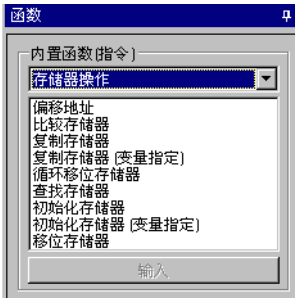
■ 矩形

项目	描述
摘要	在指定点上绘制一个矩形。当您勾选了 [图案] 复选框时，会绘制一个实心矩形。 选择并输入直线类型 (或在选择图案时选择实心图案)、颜色属性、起始和结束坐标。
格式	dsp_line (起始点 X 坐标、起始点 Y 坐标、结束点 X 坐标、结束点 Y 坐标、显示颜色闪烁 + 显示颜色、背景色闪烁 + 背景色、图案和线型)  注 释 • 当同时设置了黑色和闪烁时，背景色会变成透明的。

重 要

- 当在绘图功能中使用颜色时，请在 0 至 255 内设置颜色编号。如果设置了 E1 至 E12 且保存了脚本，会发生错误。

21.11.3 存储器操作

存储器操作	函数摘要
	偏移地址 ☞" ■ 偏移地址 " (p21-75) 设置一个地址偏移量。
	比较存储器 ☞" ■ 比较存储器 " (p21-76) 在指定位置 (偏移) 比较两个数据块，并将比较结果写入存储地址。
	复制存储器 ☞" ■ 复制存储器 " (p21-78) 在一个操作中复制存储器。
	复制存储器 (变量指定) ☞" ■ 复制存储器 (变量)" (p21-81) 在一个操作中复制存储器。可以修改源 (复制源) 地址、目标 (复制目标) 地址和地址数。
	存储器循环 ☞" ■ 循环移位存储器 " (p21-82) 按照指定字块数循环移位存储器中的数据。
	搜索存储器 ☞" ■ 搜索存储器 " (p21-84) 以块为单位执行数据搜索，并将搜索结果返回 (保存) 到指定的存储地址。
	初始化存储器 ☞" ■ 初始化存储器 " (p21-87) 立即初始化所有寄存器。
	初始化存储器 (变量指定) ☞" ■ 初始化存储器 (变量)" (p21-88) 立即初始化所有寄存器。可以修改首地址、设置数据和地址数。
	移位存储器 ☞" ■ 移位存储器 " (p21-89) 向上移位块单元。

■ 偏移地址

项目	描述
摘要	可以指定偏移地址。只能为偏移值存储地址指定临时字地址。
格式	[寄存器地址] # [偏移地址]  常量输入范围

数据类型	常量输入	
	最小	最大
Bin16	0	65535
Bin32	0	4294967295
Bin16+/-	-32768	32767
Bin32+/-	-2147483648	2147483647
BCD16	0	9999
BCD32	0	99999999

表达式示例 1：

[w:[PLC1]D0200]=[w:[PLC1]D0100]#[t:0000]

在上例中，当 [t:0000] 的值为 2 时，保存在 D0102 中的值偏移至 D0200。

表达式示例 2：

[w:[PLC1]D0100]#[t:0000]=30

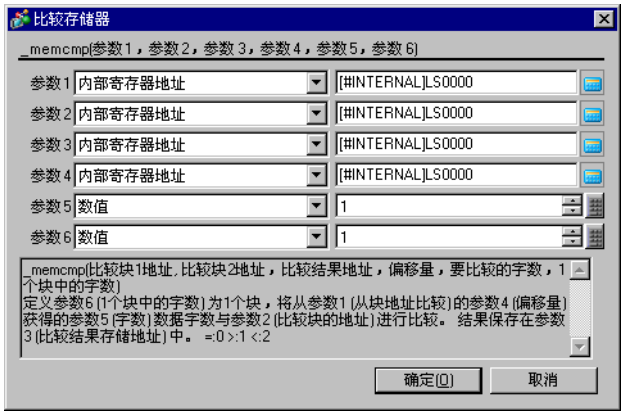
在上例中，当 [t:0000] 的值为 8 时， 30 偏移至 D0108。

重 要

- 不将偏移地址格式中使用的字地址作为 D 脚本地址。
- 对于根据偏移地址指定的寄存器中的数据，不会从外接设备中对其进行连续读取。会在 D 脚本运行时读取这些数据。当读取过程中发生错误时读数按“0”处理。此外，人机界面内部特殊继电器 LS2032 的位 12 置 ON。当正常完成数据读取时，位 12 为 OFF。
- 如果地址偏移结果超过 16 位 (最大值：65535), 位 15 及以前的位有效，位 16 及以后的位被丢弃。
- 定义地址变量时，请指定整型数组。确保整型数组足以容纳所有连续地址。否则操作将无效。如果整型变量不是数组，操作同样无效。

■ 比较存储器

项目	描述
摘要	在指定位置 (偏移) 比较两个数据块，并将比较结果写入存储地址。会将下面的值作为比较结果保存：当值等于：当目标数据大于原始数据时：当目标数据小于原始数据时：当发生错误时，会将错误状态值写入 LS9152。
格式	_memcmp ([要比较的块地址]、[比较块地址]、[比较结果存储地址], 自块起始开始的偏移 , 比较字数 , 1 个块中的字数)



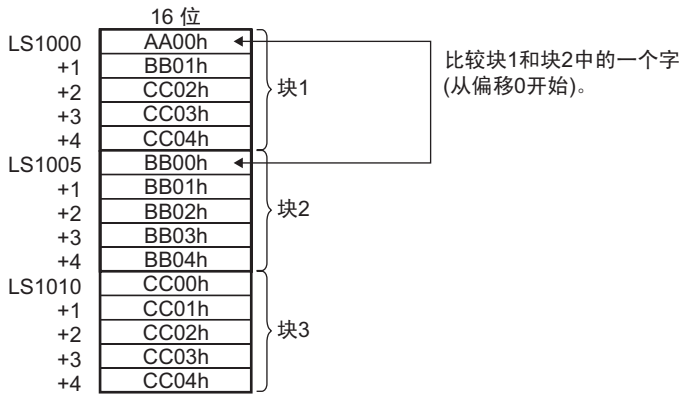
参数 1：　内部寄存器
参数 2：　内部寄存器
参数 3：　内部寄存器
参数 4：　数值 (0 至 639)、内部寄存器、临时变量
参数 5：　数值 (1 至 640)
参数 6：　数值 (1 至 640)

要保存的数据
0: 匹配
1: 比较自 < 比较至
2: 比较自 > 比较至

表达式示例 1：

_memcmp ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1005], [w:[#INTERNAL]LS0100], 0, 1, 5)

(比较块 1 和块 2 的一个字 (从偏移 0 开始) 并在 LS0100 中保存比较结果)。

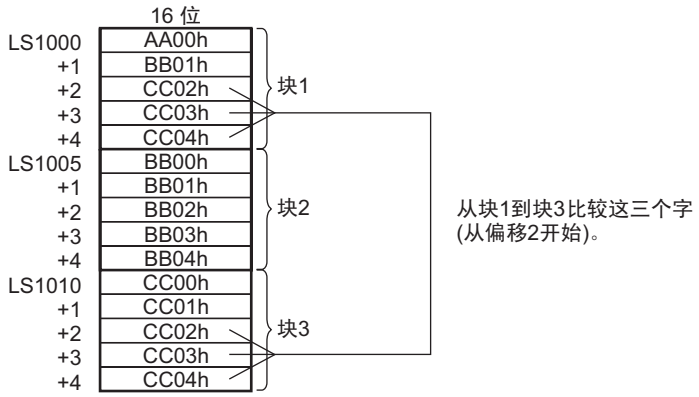


由于源值小于目标值，在 LS0100 中保存比较结果 “2”。

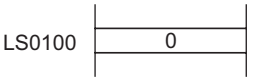
表达式示例 2：

_memcmp ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1010], [w:[#INTERNAL]LS0100], 2, 3, 5)

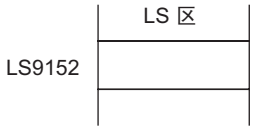
(比较块 1 和块 3 的一个字 (从偏移 2 开始) 并在 LS0100 中保存比较结果)。



由于原始值与目标值匹配，在 LS0100 保存比较结果 “0”。



错误状态



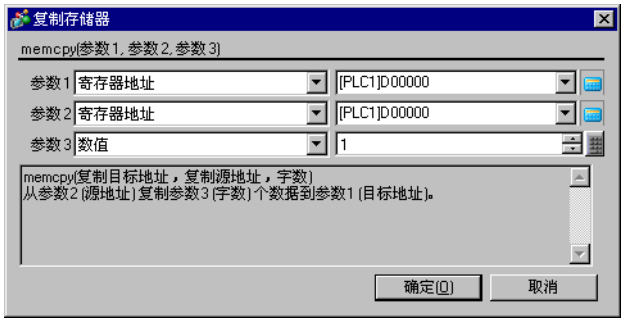
编辑器函数名称	LS 区	错误状态	原因

_memcmp ( )	LS9152	0000h	成功完成
		0001h	参数错误
		0003h	写入 / 读取错误

重 要

- 可以指定的有效 LS 寄存器范围限定为指定用户区 (LS20 至 LS2031 和 LS2096 至 LS8191)。
- 当为块偏移值指定一个大于一个块中字数的值时，该功能不起作用。
- 当要比较的字数大于一个块时，该功能不起作用。

■ 复制存储器

项目	描述
摘要	在一个操作中复制存储器。地址数量数据被复制到复制目标字地址，该字地址从源数据的第一个字地址开始。可以使用的地址数量在 1 至 640 之间。
格式	memcpy ([复制目标地址]， [复制源地址]，字数) 

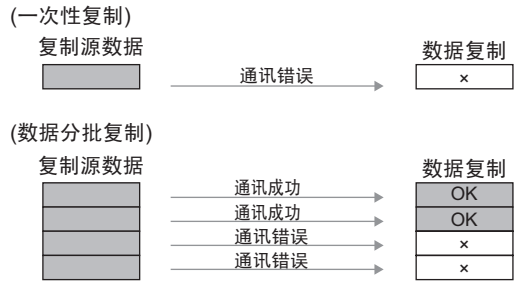
表达式示例：

memcpy ([w:[PLC1]D0200], [w:[PLC1]D0100], 10)

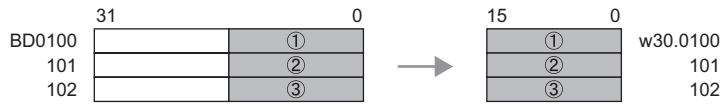
在上例中，数据从 D0100-D0109 被复制到 D0200-D0209。

重 要

- 只能在需要时从外接设备中读取源复制数据一次。如果在数据读取过程中发生通讯故障，人机界面的内部特殊继电器 LS2032 的位 12 会置 ON。当正常完成数据读取时，位 12 为 OFF。
- 可在一次操作中完成从源复制数据中读取数据和将数据写入目标，也可以将数据分成几个项目，项目的数量与用于源复制数据的地址数相等。如果在数据读取过程中发生通讯故障，数据复制的结果会根据数据是在一次操作中处理还是分几个项目处理而有如下不同：(数据写操作结果， O: 写入完成， X: 未能写入)

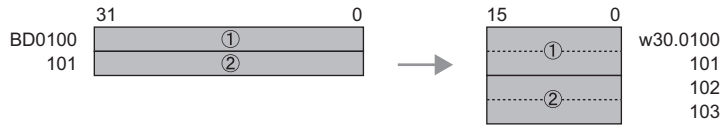


- 随着地址数量的增加，向 PLC 写入数据需要更多时间。根据地址数，可能需要 20 秒到几分钟的时间。
- 如果要写入的数据超出了指定寄存器范围，就会发生通讯错误。在这种情况下，您必须将 GP 关机后再开机，使 GP 从错误中复位。
- 当用复制存储器 (memcpy) 函数将数据写入 LS 区时，只能将数据写入用户区。不能将数据写入系统区 (LS0000 至 LS0019)、特殊区 (LS2032 至 LS2047) 或保留区 (LS2048 至 LS2095)。但是，可以从这些区中读取数据。
- 如果使用 D 脚本向 16 位寄存器复制 32 位寄存器数据，并将位长指定为 16，则只会复制低 16 位的数据。
  例如， memcpy ([w:[PLC1]w30.0100], [w:[PLC1]BD0100], 3)



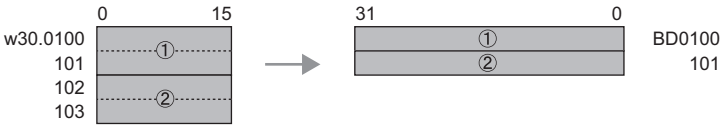
  此外，将 16 位寄存器数据复制到 32 位寄存器时，会将数据复制到低 16 位并将高 16 位置 “0”。
  例如， memcpy ([w:[PLC1]BD0100], [w:[PLC1]w30.0100], 3)

- 如果将 32 位寄存器数据复制到 16 位寄存器或将 16 位寄存器数据复制到 32 位寄存器，且脚本中指定的 D 脚本位长为 32，复制过程如下。如果其中的一个寄存器是 32 位，其他寄存器是 16 位， memcpy( ) 函数将使用 16 位作为其数据长度参数。
  例如， memcpy ([w:[PLC1]w30.0100], [w:[PLC1]BD0100], 4)

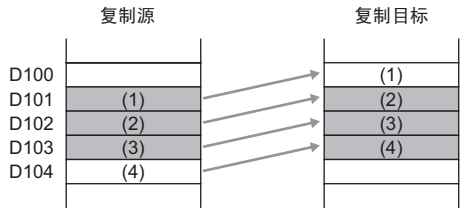


重 要

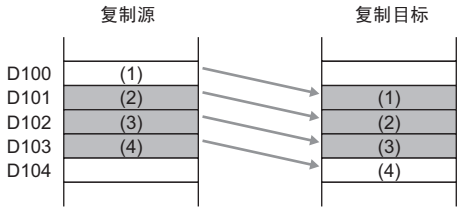
例如， memcpy ([w:[PLC1]BD0100], [w:[PLC1]w30.0100], 4)



- 如果原始和目标数据范围重叠，所有重叠数据都将按照如下方式被重写：
  例如，将 D101-D104 复制到 D100-D103
  数据被复制到编号较小的地址。

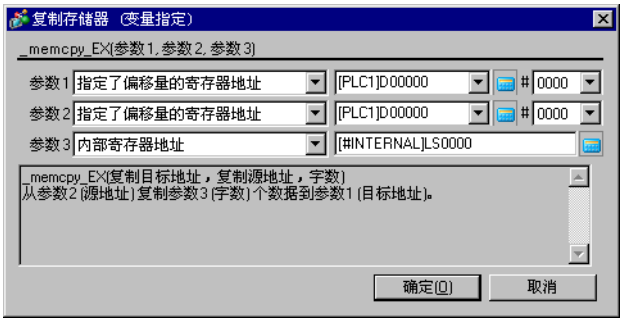


例如，将 D101-D103 复制到 D100-D104
数据被复制到编号较大的地址。



- 尽管本示例函数指定了两个地址，但不把这些地址作为 D 脚本地址。
  当使用寄存器地址进行分配时，与控制器 /PLC 的通讯会使分配值的过程有少许延迟。

■ 复制存储器 (变量)

项目	描述
摘要	在一个操作中复制存储器。将参数 3 指定的地址的数据从参数 2 指定的源字地址复制到参数 1 指定的目标字地址。 可以使用的地址数在 1 到 640 之间。使用 “_memcpy_EX” 函数，可间接指定源地址、目标地址和地址数量。
格式	_memcpy_EX([复制目标地址]、 [复制源地址]、字数) 参数 1： 寄存器地址 + 临时地址 参数 2： 寄存器地址 + 临时地址 参数 3： 数值、内部寄存器、临时地址 (参数 3 的有效范围在 1 至 640 之间。) 

表达式示例：

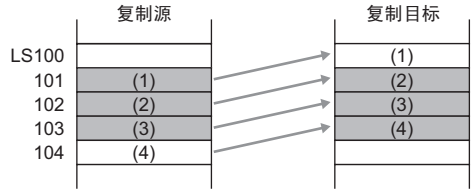
[t:0000]=10, [t:0001]=20

_memcpy_EX ([w:[#INTERNAL]LS0100]#[t:0000], [w:[PLC1]D0100]#[t:0001], 5)

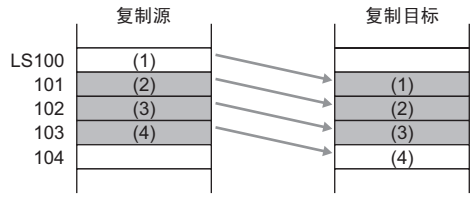
在上例中，从 D0120 中读取了数据的 5 个字并将其写入 LS110 至 LS114 中。

重 要

- 如果原始和目标数据范围重叠，所有重叠数据都将按照如下方式被重写：
  例如，将 LS101-LS104 复制到 LS100-LS103 中
  数据被复制到编号较小的地址。

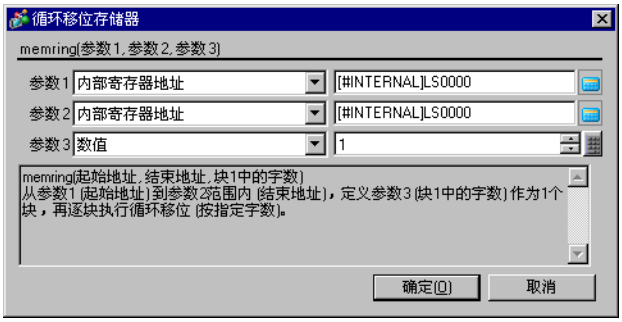


例如，将 LS101-LS103 复制到 LS100-LS104 中
数据被复制到编号较大的地址。



■ 循环移位存储器

项目	描述
摘要	以块为单位循环移位存储器中的数据。 以块为单位 (按指定字数) 在起始和结束地址之间执行循环移位。 当发生错误时，会将错误状态写入 LS9150。
格式	memring ([起始地址]、 [结束地址]、 1 个块中的字数)



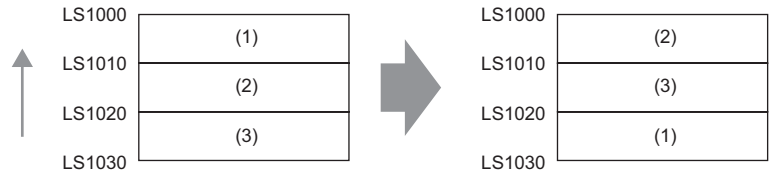
参数 1：　内部寄存器
参数 2：　内部寄存器
参数 3：　数值 (1 至 640)
- 当参数 1 小于参数 2 时 (P1<P2)，块数据向上移位。
- 当参数 1 大于参数 2 时 (P1>P2)，块数据向下移位。

重 要

• 请务必将起始地址和结束地址设置为同类型寄存器 (LS 或 USR)。

表达式示例 1：

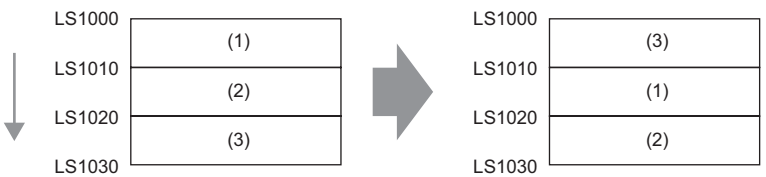
memring ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1030], 10)
(当参数 1 大于参数 2 时 (< P2))



数据以 10 个字块为单位向上移动。

表达式示例 2：

memring ([w:[#INTERNAL]LS1030], [w:[#INTERNAL]LS1000], 10)
(当参数 1 大于参数 2 时 (> P2))

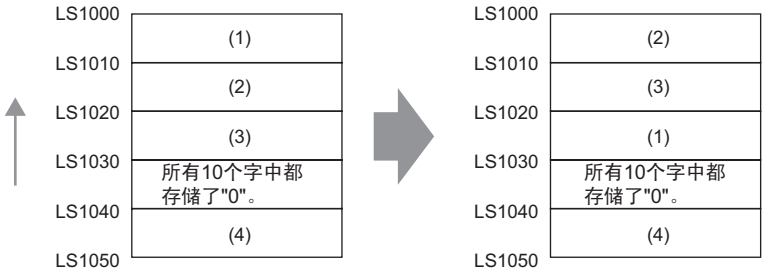


数据以 10 个字块为单位向下移动。

表达式示例 3：

memring ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1050], 10)

(当该范围包含所有字均为 “0” 的块时。)

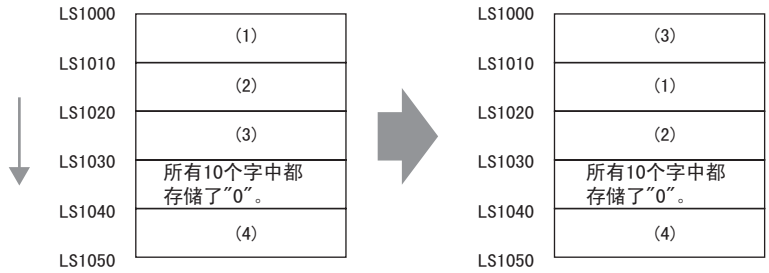


数据仅以 10 个字块为单位向上移动，从起始块到 “0” 数据块。如果在 “0” 数据块后还存在数据，那么这些数据会被忽略。

表达式示例 4：

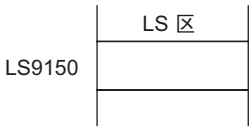
memring ([w:[#INTERNAL]LS1050], [w:[#INTERNAL]LS1000], 10)

(当范围内存在 “0” 数据块时。)



数据仅以 10 个字块为单位向下移动，从起始块到 “0” 数据块。如果在 “0” 数据块后还存在数据，那么这些数据会被忽略。

错误状态



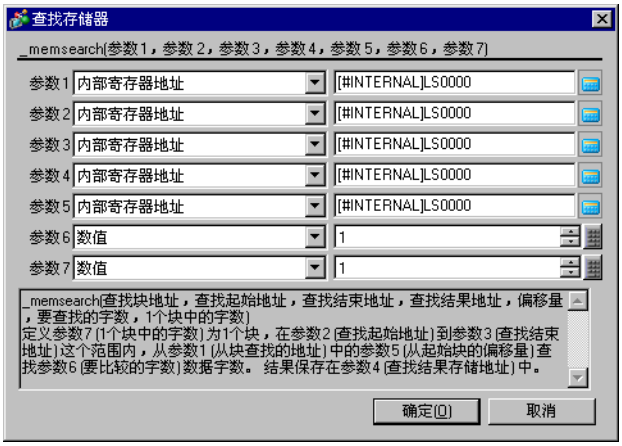
编辑器函数名称	LS 区	错误状态	原因
memring ( )	LS9150	0000h	成功完成
		0001h	参数错误
		0003h	写入 / 读取错误

重 要

- 所需的处理时间与开始和结束地址指定的范围成比例。指定的范围越大，处理时间越长。直到处理完成后才刷新部件。
- 可以指定的有效 LS 寄存器范围限定为指定用户区 (LS20 至 LS2031 和 LS2096 至 LS8191)。

■ 搜索存储器

项目	描述
摘要	以块为单位执行数据搜索，搜索从指定范围内的第一个项目开始。从指定的 (偏移) 块开始比较数据块，并将搜索结果返回 (保存) 到指定的存储地址。当发现匹配块时，就保存该块的偏移值 (1 或更高)。未发现匹配块时，就保存 “FFFFh”。当发生错误时，会将错误状态值写入 LS9153。
格式	_memsearch([要搜索的块地址]、[搜索起始地址]、[搜索结束地址]、[搜索结果存储地址]、自起始块的偏移、比较字数、1 个块中的字数)



参数 1：　内部寄存器
参数 2：　内部寄存器
参数 3：　内部寄存器
参数 4：　内部寄存器
参数 5：　数值 (0 至 639)、内部寄存器、临时变量
参数 6：　数值 (1 至 640)
参数 7：　数值 (1 至 640)

要写入的数据
当有匹配块时：块的偏移值 (“1” 或更高)
当没有匹配块时："FFFFh"

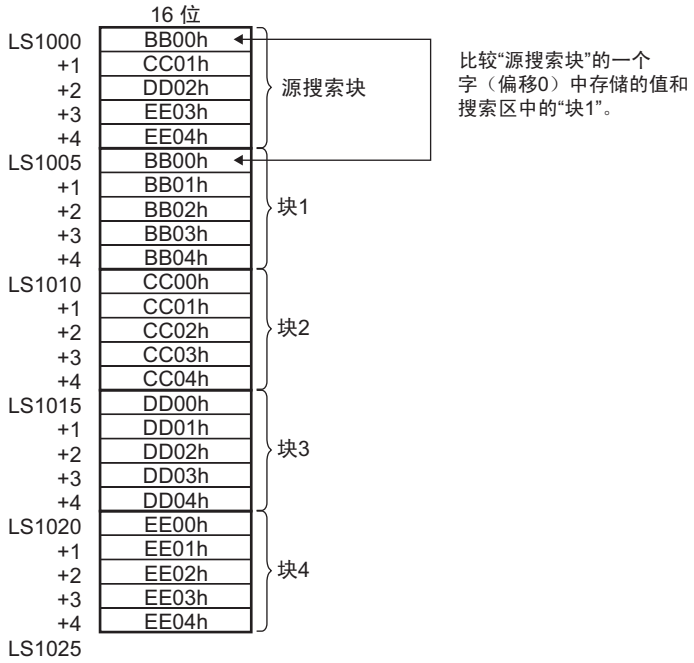
重 要

- 请务必将搜索起始地址和搜索结束地址设置为同类型寄存器 (LS 或 USR)。但是，可将 [要搜索的块地址] 和 [搜索结果存储地址] 设置为内部寄存器。
- 确保 [参数 2] 小于 [参数 3]。否则，会发生错误。

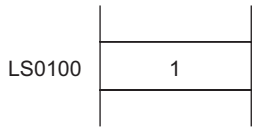
表达式示例 1：

_memsearch ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1005], [w:[#INTERNAL]LS1025],[w:[#INTERNAL]LS0100], 0, 1, 5)

(从 LS1005 至 LS1025 中搜索具有相同值的块。从源搜索块的偏移 0 开始，将搜索结果保存在 LS0100 中。)



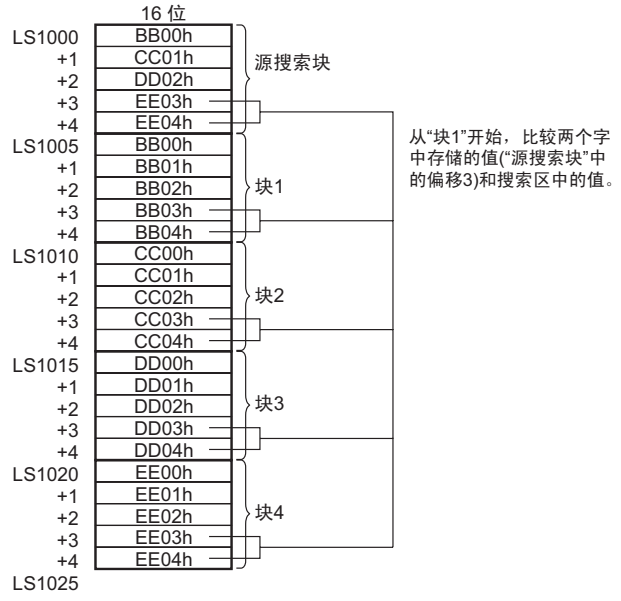
本例中，“块 1”的值就与 “源搜索块”的值匹配。因此，在 LS0100 中保存搜索结果 “1”。



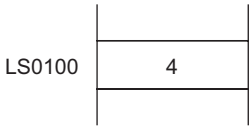
表达式示例 2：

_memsearch ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1005],
[w:[#INTERNAL]LS1025],
[w:[#INTERNAL]LS0100], 3, 2, 5)

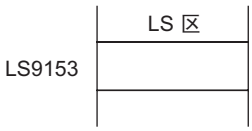
(从 LS1005 至 LS1025 中搜索具有相同值的块。用两个字，从偏移 3 开始，并将结果保存在 LS0100 中。)



本例中，“块 4”的值就与 “源搜索块”的值匹配。因此，在 LS0100 中保存搜索结果 “4”。



错误状态



编辑器函数名称	LS 区	错误状态	原因
_memsearch ( )	LS9153	0000h	成功完成
		0001h	参数错误
		0003h	写入 / 读取错误

重 要

- 所需的处理时间与开始和结束地址指定的范围成比例。指定的范围越大，处理时间越长。直到处理完成后才刷新部件。
- 可以指定的有效 LS 寄存器范围限定为指定用户区 (LS20 至 LS2031 和 LS2096 至 LS8191)。

■ 初始化存储器

项目	描述
摘要	立即初始化所有寄存器。地址编号的设置数据取自已设置的字地址。地址数量的有效范围在 1 至 640 之间。
格式	memset ([写入地址]、写入数据、字数)

表达式示例：

memset ([w:[PLC 1]D0100], 0, 10)

在上面的例子中，将 D0100 至 D0109 的地址置 “0”。

重 要

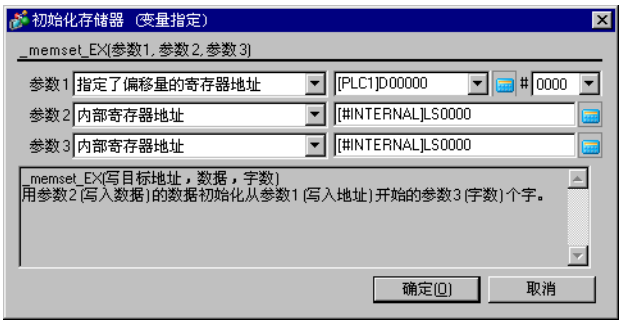
- 随着地址数量的增加，向 PLC 写入数据需要更多时间。根据地址数，可能需要 20 秒到几分钟的时间。
- 如果要写入的数据超出了指定寄存器范围，就会发生通讯错误。在这种情况下，您必须将 GP 关机后再开机，使 GP 从错误中复位。
- 尽管该函数会指定地址，但它们不被视作 D 脚本地址。
- 当用存储器复位 (memset) 函数向 LS 区写入数据时，只能将数据写入用户区。不能将数据写入系统数据区 (LS0000 至 LS0019)、特殊区 (LS2032 至 LS2047) 或保留区 (LS2048 至 LS2095)。
- 当分配操作使用寄存器地址时，由于 GP 至 PLC 的传输时间问题，不会立即分配写入值。

  例如：

  ```
  memset ([w:D0100], 0, 10)        // 初始化 "D100 至 D109" 为 0
  [ w:[PLC1]D200]=[w:[PLC1]D100]// 替换 D100 至 D200
  ```

  在这种情况下，被写入 D100 的运算结果 0 不也分配给 D200。

■ 初始化存储器 (变量)

项目	描述
摘要	立即初始化所有寄存器。将从参数 1 指定的设置字地址开始到参数 3 指定的地址设置为参数 2 指定的设置数据。地址数量的有效范围在 1 至 640 之间。可以间接分配写入地址、写入数据和地址数量。
格式	_memset_EX ([写入地址]、写入数据、字数)  参数 1：寄存器地址 + 临时地址 参数 2：数值、内部寄存器、临时地址 (参数 2 的有效范围对十进制来说为 0 至 65535，对十六进制来说为 0 至 FFFF) 参数 3：数值、内部寄存器、临时地址 (参数 3 的有效范围在 1 至 640 之间。)

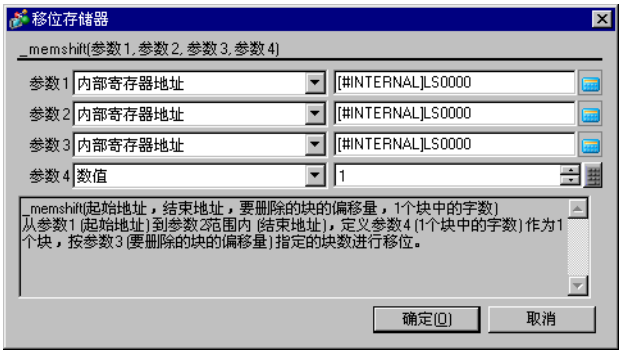
表达式示例：

[t:0000]=10
[w:[#INTERNAL]LS0050]=0
[w:[#INTERNAL]LS0051]=5
_memset_EX ([w:[#INTERNAL]LS0100]#[t:0000], [w:[#INTERNAL]LS0050], [w:[#INTERNAL]LS0051])

在上面的例子中，“0”将被写入从 LS0100 至 LS0114 这 5 个字中。

■ 移位存储器

项目	描述
摘要	删除指定块并将如下数据块上移。要删除的块使用偏移加以指定。当发生错误时，会将错误状态写入 LS9151。
格式	_memshift ([起始地址]、 [结束地址]、要删除的块偏移、 1 个块中的字数)



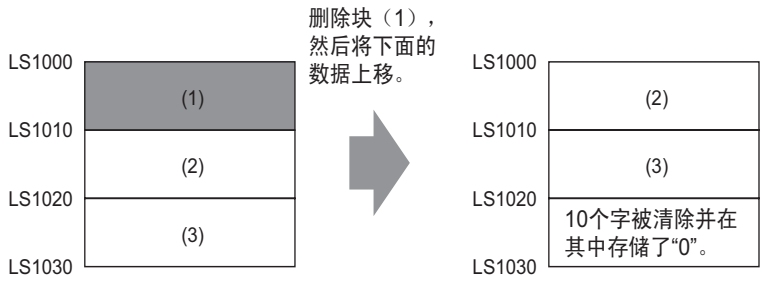
参数 1：　内部寄存器
参数 2：　内部寄存器
参数 3：　数值 (1 至 65,535)、内部寄存器、临时变量
参数 4：　数值 (1 至 640)

重 要

- 请务必将起始地址和结束地址设置为同类型寄存器 (LS 或 USR)。
- 确保 [参数 1] 小于 [参数 2]。否则，会发生错误。

表达式示例 1：

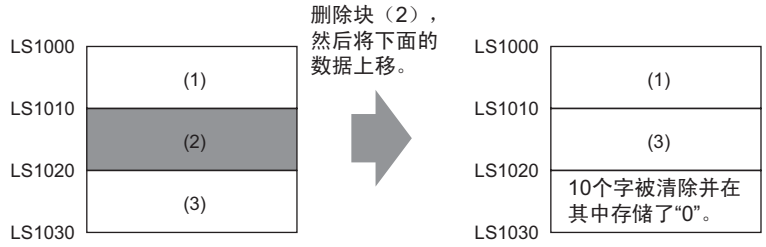
_memshift ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1030], 1, 10)



数据以块为单位上移 (1 个块 =10 个字)，最后一个块 (10 个字) 被清零。

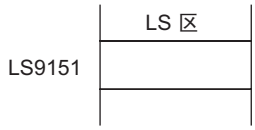
表达式示例 2：

_memshift ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1030], 2, 10)



数据从偏移 2 位置开始以块为单位上移 (1 个块 =10 个字)，最后一个块 (10 个字) 被清零。

错误状态



编辑器函数名称	LS 区	错误状态	原因
_memshift ( )	LS9151	0000h	成功完成
		0001h	参数错误
		0003h	写入 / 读取错误

重 要

- 所需的处理时间与开始和结束地址指定的范围成比例。指定的范围越大，处理时间越长。直到处理完成后才刷新部件。
- 如果将一个超出了指定开始和结束地址范围的值指定为要删除块的偏移量，该函数不能正常运行。
- 可以指定的有效 LS 寄存器范围限定为指定用户区 (LS20 至 LS2031 和 LS2096 至 LS8191)。

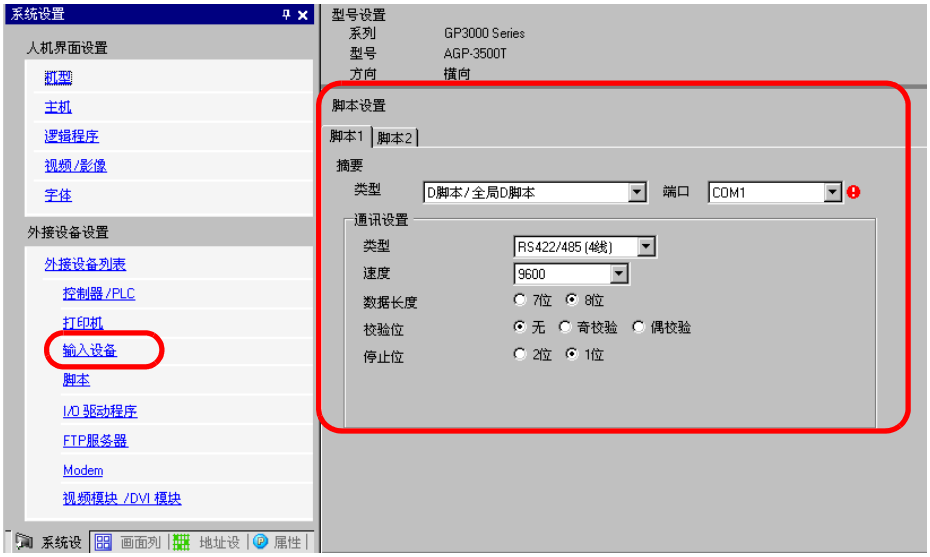
21.11.4 串口操作

串口操作	函数摘要
	标签设置 ☞" ■ 标签设置 " (p21-93) 从控制、状态、接收数据计数、接收功能和发送功能中设置。
	接收 ☞" ■ 接收 " (p21-95) 从指定串口 (COM1 或 COM2) 读取已接收数据。
	发送 ☞" ■ 发送 " (p21-96) 写入指定串口 (COM1 或 COM2)。
	扩展接收 ☞" ■ 扩展接收 " (p21-97) 从指定串口 (COM1 或 COM2) 读取已接收数据。 只能在扩展脚本中使用它。
	扩展发送 ☞" ■ 扩展发送 " (p21-98) 写入指定串口 (COM1 或 COM2)。 只能在扩展脚本中使用它。
	待机接收功能 ☞" ■ 待机接收功能 " (p21-99) 保持在待机接收模式中，直到它接收到指定文本。 只能在扩展脚本中使用它。
	待机功能 ☞" ■ 待机功能 " (p21-100) 系统在指定时间内处于等待状态。 只能在扩展脚本中使用它。

重 要

- 可在 D 脚本 / 全局 D 脚本中方便地包含标签设置、发送和接收。
- 要与 D 脚本 / 全局 D 脚本通讯，请进行如下的脚本设置。如果未指定脚本设置，它们就不能执行。

[D 脚本 / 全局 D 脚本 I/O 步骤]
在 [系统设置] 窗口，点击 [脚本]。
将 [类型] 设置为 [D 脚本 / 全局 D 脚本]。



有两个标签可以设置脚本。上面示例使用 [脚本 1]。
将 [串口] 设置为 COM1 或 COM2，并设置 [通讯设置] 以匹配扩展 SIO。

- 当创建比串口操作具备更先进功能的通讯程序时，我们建议使用 [扩展脚本]。参见以下示例，了解如何使用扩展脚本的信息，
☞"21.5 与不支持的外接设备通讯 " (p21-21)

■ 标签设置

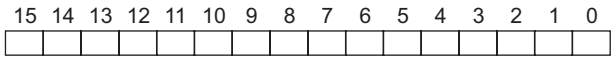
控制

项目	描述
摘要	该控制变量用于清除发送缓冲器、接收缓冲器和错误状态。该控制变量是只写的。
格式	当指定位时：[c:EXT_SIO_CTRL**]**: 00 至 15) 当指定字时：[c:EXT_SIO_CTRL]

◆ 表达式示例

当指定位时：[c:EXT_SIO_CTRL00] = 1
当指定字时：[c:EXT_SIO_CTRL] = 0x0007

◆ EXT_SIO_CTRL



位	内容
15	保留
14	
13	
12	
11	
10	
9	
8	
7	
6	
5	
4	
3	1: 清除接收超时
2	1: 清除错误
1	1: 清除接收缓冲器
0	1: 清除发送缓冲器

注 释

- 如果选择字，且同时置位两个以上的位，按如下顺序执行处理：清除错误 -> 清除接收缓冲器 -> 清除发送缓冲器

状态

项目	描述
摘要	状态包含如下信息。 该状态变量为只写。
格式	当指定位时：[s:EXT_SIO_STAT**](** ： 00 至 15) 当指定字时：[s:EXT_SIO_STAT]

◆ 表达式示例

当指定位时：如果 ([s:EXT_SIO_STAT 00] == 1)
当指定字时：if (([s:EXT_SIO_STAT] & 0x0001) <> 0)

◆ EXT_SIO_STAT 的内容

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

位	内容
15	0: 无D脚本/全局D脚本 1: D脚本/全局D脚本已经存在
14	0: 无扩展脚本 1: 扩展脚本已经存在
13	保留
12	
11	
10	
9	
8	
7	
6	
5	
4	0: 正常 1: 接收超时
3	0: 正常 1: 接收错误
2	0: 无接收数据 1: 接收数据已经存在
1	0: 正常 1: 发送错误
0	0: 数据在发送缓冲器中已经存在 1: 发送缓冲器为空

注 释

- 可能会在将来为保留位分配数据。因此，请务必只检查必需的位。
- 存在两种类型的传输错误：传输超时错误和传输缓冲器已满错误。当发生这两类错误中的任何一类时，传输错误位都会置 ON。传输超时时间为 5 秒钟。
- 有四种类型的接收错误：奇偶校验错误、溢位错误、帧错误和溢出错误。当发生上述这四个错误中的任一错误时，接收错误位就会置 ON。
- 如果检测到传输错误，发送数据会保持在传输缓冲器中。如果没有检测到传输错误，发送数据会从传输缓冲器中发出。
- 当使用串行接口 COM2(它是 RS-422) 时，就不能检测 CS(CTS) 信号。因此，将不能检测到电缆连接的断开。

已接收数据大小

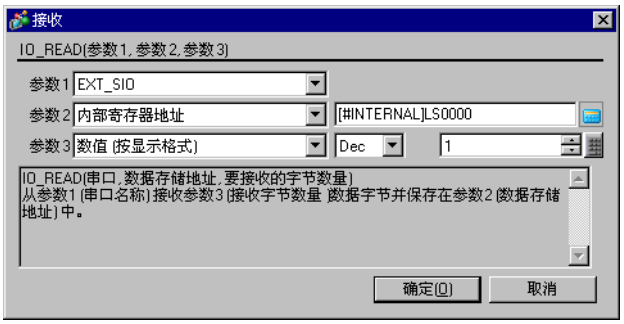
项目	描述
摘要	显示此时已经接收的数据量 (字节数)。已接收数据大小为只读。
格式	[r:EXT_SIO_RECV]

重 要

- 已接收数据量 (字节数) 的标签名称
  使用 GP-PRO/PB III V.6.0 和更早的版本，为已接收数据大小指定的标签名称是 [r: EXT_SIO_RCV]。但是，由于功能是相同的，您无需修订该描述，无论选择了 [r: EXT_SIO_RCV] 还是 [r: EXT_SIO_RECV] 表达式。

■ 接收

项目	描述
摘要	当从扩展串口中读取已接收数据时，如下所示编写语句。
格式	IO_READ ([p:EXT_SIO]，数据存储地址、接收字节数)



参数 1：　EXT_SIO
参数 2：　内部寄存器
参数 3：　数值

表达式示例：

IO_READ ([p:EXT_SIO], [w:[#INTERNAL]LS0100], 10)

上例中，在 LS0100 中保存已接收字节数。从 LS0101 开始保存 10 个字节的数据。如下图片显示保存的已接收数据。

注 释

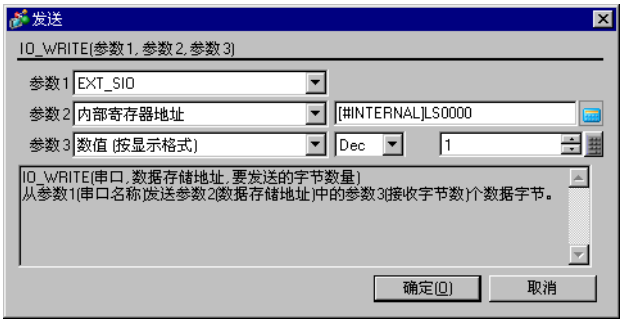
- 数据接收期间传输字节的最大数量是 2011。以 1 字节为单位向每个字地址写入数据。

LS0100	接收数据字节		... 10 字节
LS0101	00	字节1	
LS0102	00	字节2	
LS0103	00	字节3	
LS0104	00	字节4	
LS0105	00	字节5	
LS0106	00	字节6	
LS0107	00	字节7	
LS0108	00	字节8	
LS0109	00	字节9	
LS0110	00	字节10	

接收的数据存储方式

■ 发送

项目	描述
摘要	将数据写至扩展串口时，如下所示书写语句。
格式	IO_WRITE ([p:EXT_SIO]，数据存储地址，发送字节数)



参数 1：　EXT_SIO
参数 2：　内部寄存器
参数 3：　数值

表达式示例：

IO_WRITE ([p:EXT_SIO], [w:[#INTERNAL]LS0100], 10)

上例中，发送从 LS0100 开始的 10 个字节的数据。下图显示保存的已发送数据。

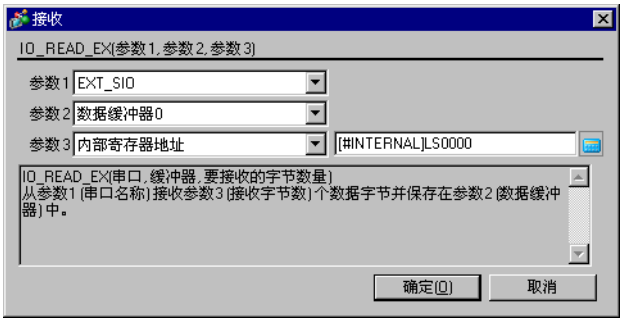
注 释

- 接收数据时传输字节的最大数是 2012。
- 和发送缓冲器的 LS 寄存器一样，向每个字地址写入单字节数据。

LS0100	00	字节1
LS0101	00	字节2
LS0102	00	字节3
LS0103	00	字节4
LS0104	00	字节5
LS0105	00	字节6
LS0106	00	字节7
LS0107	00	字节8
LS0108	00	字节9
LS0109	00	字节10

发送的数据存储方式

■ 扩展接收

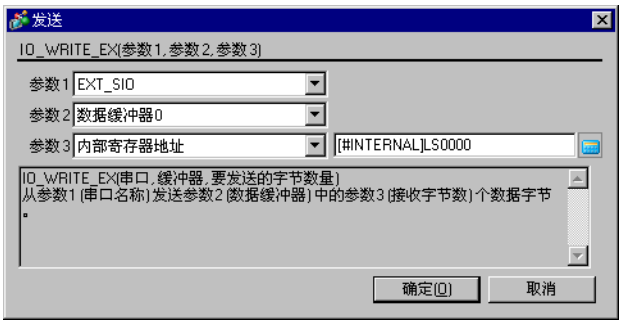
项目	描述
摘要	从扩展串口中接收数据，数据量在接收数据大小 (字节) 中指定，并将它保存在数据缓冲器中。从扩展串口中接收参数 3 指定的字节数，并将其保存在参数 2 指定的数据缓冲器中。 只能在扩展脚本中使用它。
格式	IO_READ_EX ([p:EXT_SIO]，数据缓冲器，接收字节数)  参数 1：[p:EXT_SIO] 参数 2：数据缓冲器 参数 3：数值、内部寄存器、临时地址 (参数 3 的有效范围在 1 至 1024 之间。)

表达式示例：

IO_READ_EX ([p:EXT_SIO], databuf1, 10)

上例中，在 “数据缓冲器 1”中接收和保存扩展串口接收的 10 字节的数据。

■ 扩展发送

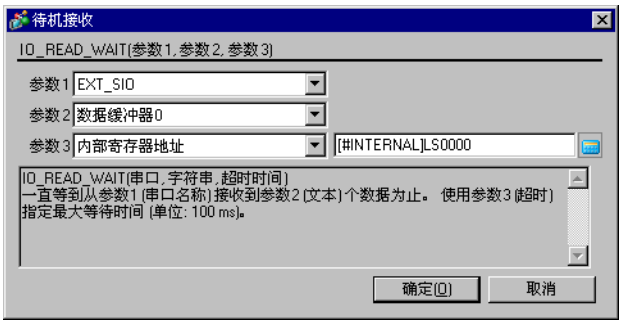
项目	描述
摘要	根据发送字节数大小用扩展串口发送数据缓冲器中的数据。按参数 3 指定的长度从扩展串口中发送参数 2 指定的数据缓冲器的内容。只能在扩展脚本中使用它。
格式	IO_WRITE_EX ([p:EXT_SIO]，数据缓冲器，发送字节数)  参数 1：[p:EXT_SIO] 参数 2：数据缓冲器 参数 3：数值、内部寄存器、临时地址 (参数 3 的有效范围在 1 至 1024 之间。)

表达式示例：

IO_WRITE_EX ([p:EXT_SIO], databuf0, 10)

在上面的例子中，从扩展串口中发送 “databuf0”中的 10 字节数据。

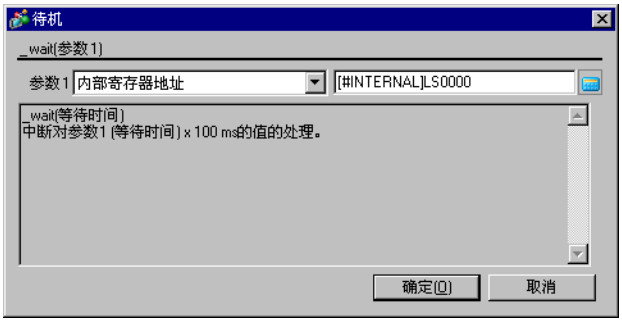
■ 待机接收功能

项目	描述
摘要	保持在待机接收模式中，直到它接收到指定文本。在达到超时时间后，状态的位 4(接收超时错误)[s：EXT_SIO_STAT] 被置位。设置超时持续时间时可按 100ms 增加。 系统在接收到字符串或参数 2 指定的字符代码前一直处于待机接收模式。用参数 3 配置超时时间。 只能在扩展脚本中使用它。
格式	IO_READ_WAIT ([p:EXT_SIO]，文本，超时)  参数 1：[p:EXT_IO] 参数 2：数值、文本、数据缓冲器 参数 3：数值、内部寄存器、临时地址 (参数 3 的有效范围在 1 至 600 之间。)

重 要

- 在接收到指定文本前不能使用已接收数据。 (否则，数据将被丢弃。)
- 最多可以指定 128 个字符 (字节)。需要注意的是，当指定了超过范围的字符串时，就不能成功执行待机接收操作。

■ 待机功能

项目	描述
摘要	系统在指定时间内处于等待状态。该时间可按 100ms 的增量进行配置。 只能在扩展脚本中使用它。
格式	_wait(等待时间)  参数 1：　　内部寄存器、临时地址、数值 (参数 1 的有效范围为 1 至 600)

表达式示例：

_wait (10)

在上例中，系统等待 1 秒钟。

21.11.5 CF 卡文件操作 /USB 文件操作

<table>
<tr><th>CF 卡文件操作</th><th>函数摘要</th></tr>
<tr><td rowspan="7"> 


</td><td>标签设置
☞ " ■ 标签设置 " (p21-102)
从列出的文件数、已读字节数和 CF 卡 /USB 存储器错误状态中设置。</td></tr>
<tr><td>写入文件
☞ " ■ 写入文件 " (p21-111)
可以选择如下三种模式的任意一种：</td></tr>
<tr><td>文件名更改
☞ " ■ 文件名更改 " (p21-115)
修改文件名。</td></tr>
<tr><td>读 CSV 文件
☞ " ■ 读取 CSV 文件 " (p21-117)
从 CSV 文件中以元素为单位读取数据并将它写入字地址中。</td></tr>
<tr><td>读取文件
☞ " ■ 读取文件 " (p21-119)
在指定偏移后读取文件中指定字节数的数据并将它写入目标地址。</td></tr>
<tr><td>输出文件列表
☞ " ■ 输出文件列表 " (p21-121)
指定文件夹中存在的文件列表被写入内部寄存器。</td></tr>
<tr><td>删除文件
☞ " ■ 删除文件 " (p21-123)
删除文件。</td></tr>
</table>

■ 标签设置

下面是 CF 卡 /USB 存储器状态的可能状态值。

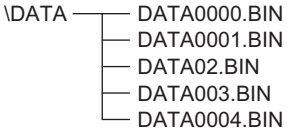
状态名	标签名称	描述
列出的文件	[s:CF_FILELIST_NUM] [s:USB_FILELIST_NUM]	执行文件列表输出函数 “_CF_dir ( )” 或 “_USB _dir ( )” 时保存实际列出的文件数。
已读取字节数	[s:CF_READ_NUM] [s:USB_READ_NUM]	当执行了文件读取函数 “_CF_read ( )” 时保存可以实际读取的字节数。
CF 卡 /USB 存储器错误状态	[s:CF_ERR_STAT] [s:USB_ERR_STAT]	保存访问 CF 卡或 USB 存储器时产生的错误状态。

列出的文件

当执行了文件列表输出函数 “_CF_dir ( )” 或 “_USB _dir ( )” 时，会将在 LS 区实际写入的文件列表数保存在 “列出的文件 [s:CF_FILELIST_NUM]/ [s:USB_FILELIST_NUM]” 中。

◆ 使用示例

```
_CF_dir ("\DATA\*.*", [w:[#INTERNAL]LS0100], 10, 0)
[w:LS0200] = [s:CF_FILELIST_NUM]
```



当获取 10 个文件的文件列表而指定文件夹只包含 5 个文件时，在 [s:CF_FILELIST_NUM] 中会保存 “5”。

重 要

- 当未写入文件时，会将指定文件夹中包含的总文件数写入 [s:CF_FILELIST_NUM] 中。

已读取字节数

当执行文件读取函数 “_CF_read ()” 或 “_USB _read ()” 时，在 “读出字节 [s:CF_READ_NUM] / [s:USB_READ_NUM] 中保存实际读出的字节数。

◆ 使用示例

```
_CF_read ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 16, 16)
[w:[#INTERNAL]LS0200] = [s:CF_READ_NUM]
```

当试图读取 16 个字节但只成功读出 12 个字节时，
在 [s:CF_READ_NUM] 中会保存 “12”。

CF 卡 /USB 存储器错误状态

保存访问 CF 卡或 USB 存储器时产生的错误状态。

位位置	错误名称	描述
15	保留	保留
14		
13		
12		
11		
10		
9		
8		
7		
6	文件重命名错误	• CF 卡 /USB 存储器在执行过程中被拔除。 • 指定文件不存在。
5	文件删除错误	• CF 卡 /USB 存储器在执行过程中被拔除。 • 指定文件不存在。 • 试图删除一个具有只读属性的文件。
4	文件写入错误	• CF 卡 /USB 存储器在执行过程中被拔除。 • 超出了 CF 卡 /USB 存储器的可用空间。 • 试图向一个具有只读属性的文件写入数据。 • 试图 “覆盖” 不存在的文件。
3	文件读取错误	• CF 卡 /USB 存储器在执行过程中被拔除。 • 指定文件不存在。
2	文件列表错误	• CF 卡 /USB 存储器在执行过程中被拔除。 • 指定文件夹不存在。
1	CF 卡 /USB 存储器错误	• CF 卡 /USB 存储器无效。 • 插入的介质不是 CF 卡。
0	CF/USB 存储器缺少 CF 卡	• 未插入 CF 卡 /USB 存储器。 • 舱盖打开。

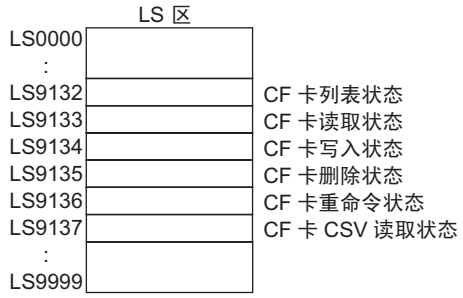
- 即使在发生 CF 卡 /USB 存储器错误时，处理仍继续。务必编写脚本，令其在您使用 CF 卡 /USB 存储器的文件操作功能时进行错误检查。

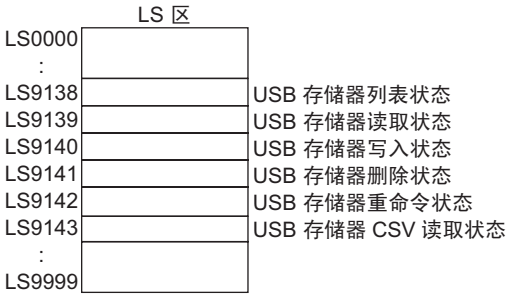
  例如：

```
_CF_dir ("\DATA\*.*", [w:[#INTERNAL]LS0100], 2, 1) 输出一个文件列表。
if ([s:CF_ERR_STAT02] <> 0)        // 检查错误状态。
{
    set([b:[#INTERNAL]LS005000]) // 为错误显示设置位地址
}
endif
```

◆ CF 卡 /USB 存储器错误详细状态保存区

当发生错误时将对每个位进行置位。您可以通过设置详细状态查看到底是什么原因导致错误。在每个函数中，详细状态保存在扩展系统区的 LS9132 至 LS9137(对 USB 存储器来说是 LS9138 至 LS9143)。这些区仅供读入。





◆ 每个函数的错误列表

编辑器函数名称		错误状态	原因
_CF_dir ( )	LS9132	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称时发生的错误)
		0012h	文件名 (路径名) 错误
		0018h	LS 区写入范围错误
		0020h	无 CF 卡
		0021h	CF 卡无效
		0100h	目录打开错误
_CF_read ( )	LS9133	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称 / 文件名时发生的错误)
		0011h	LS 区读取范围错误
		0012h	文件名 (路径名) 错误
		0018h	LS 区写入范围错误
		0020h	无 CF 卡
		0021h	CF 卡无效
		0101h	文件查找错误 (偏移错误)
		0102h	已读取字节数错误
		0110h	文件创建 (打开) 错误

编辑器函数名称		错误状态	原因
_CF_write ( )	LS9134	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称 / 文件名时发生的错误)
		0011h	LS 区读取范围错误
		0012h	文件名 (路径名) 错误
		0020h	无 CF 卡
		0021h	CF 卡无效
		0101h	文件查找错误 (偏移错误)
		0104h	文件夹创建错误
		0108h	写入模式错误
		0110h	文件创建 (打开) 错误
		0111h	文件写入错误 (例如， CF 卡上的空间不足)
_CF_delete ( )	LS9135	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称 / 文件名时发生的错误)
		0011h	LS 区读取范围错误
		0012h	文件名 (路径名) 错误
		0020h	无 CF 卡
		0021h	CF 卡无效
		0112h	文件删除错误 (例如，指定的文件不存在。指定文件是只读文件。)
_CF_rename ( )	LS9136	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称 / 文件名时发生的错误)
		0011h	LS 区读取范围错误
		0012h	文件名 (路径名) 错误
		0020h	无 CF 卡
		0021h	CF 卡无效
		0114h	文件重命名错误 (例如，指定的文件不存在。文件名已经存在。)
_CF_read_csv ( )	LS9137	0001h	参数错误
		0002h	CF 卡错误 (无 CF 卡、打开文件错误、文件读取错误)
		0003h	写入错误

编辑器函数名称		错误状态	原因
__USB_dir ( )	LS9138	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称时发生的错误)
		0012h	文件名 (路径名) 错误
		0018h	LS 区写入范围错误
		0020h	无 USB 存储器
		0021h	USB 存储器无效
		0100h	目录打开错误
__USB_read ( )	LS9139	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称 / 文件名时发生的错误)
		0011h	LS 区读取范围错误
		0012h	文件名 (路径名) 错误
		0018h	LS 区写入范围错误
		0020h	无 USB 存储器
		0021h	USB 存储器无效
		0101h	文件查找错误 (偏移错误)
		0102h	已读取字节数错误
		0110h	文件创建 (打开) 错误
__USB_write ( )	LS9140	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称 / 文件名时发生的错误)
		0011h	LS 区读取范围错误
		0012h	文件名 (路径名) 错误
		0020h	无 USB 存储器
		0021h	USB 存储器无效
		0101h	文件查找错误 (偏移错误)
		0104h	文件夹创建错误
		0108h	写入模式错误
		0110h	文件创建 (打开) 错误
		0111h	文件写入错误 (例如：USB 存储器上的空间不足)

编辑器函数名称		错误状态	原因
__USB_delete ( )	LS9141	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称 / 文件名时发生的错误)
		0011h	LS 区读取范围错误
		0012h	文件名 (路径名) 错误
		0020h	无 USB 存储器
		0021h	USB 存储器无效
		0112h	文件删除错误 (例如，指定的文件不存在。指定文件是只读文件。)
__USB_rename ( )	LS9142	0010h	无效的 D 脚本数据 (检索用固定字符串指定的文件夹名称 / 文件名时发生的错误)
		0011h	LS 区读取范围错误
		0012h	文件名 (路径名) 错误
		0020h	无 USB 存储器
		0021h	USB 存储器无效
		0114h	文件重命名错误 (例如，指定的文件不存在。文件名已经存在。)
__USB_read_csv ( )	LS9143	0001h	参数错误
		0002h	USB 存储器错误 (无 USB 存储器、文件打开错误、文件读取错误)
		0003h	写入错误

◆ 数据存储模式

当执行文件读取 / 文件写入函数，从 / 向寄存器地址读取 / 写入数据时，可以指定已写入 (读取) 数据的存储顺序。

在 LS9130 中设置数据存储模式可以更改存储顺序。可以从如下四种模式中任意选择一种：0, 1, 2 或 3。

注 释

- 使用以下命令来参考 LS9130。
  _CF_write()　　CF 文件操作：写入文件
  _CF_read()　　CF 文件操作：读取文件
  _CF_read_csv()　CF 文件操作：读 CSV 文件
  _USB_write()　　USB 文件操作：写入文件
  _USB_read()　　USB 文件操作：读取文件
  _USB_read_csv()　USB 文件操作：读 CSV 文件
  IO_WRITE([p:PRN],...)打印机操作：发送
- 当写入或读取寄存器地址时，可以使用以下函数而不是 LS9130 存储模式，与 [系统设置] 窗口的 [控制器 /PLC] 页中的 [文本数据模式] 属性进行交互。
  _CF_dir()　　CF 文件操作：输出文件列表
  _USB_dir()　　USB 文件操作：输出文件列表

- 模式 0

当使用文件读取函数向寄存器地址中写入字符串 “ABCDEFG” 时

[w:[#INTERNAL]LS9130] = 0

_CF_read ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 7)

- 当寄存器地址长度为 16 位时

LS0100	‘A’	‘B’
LS0101	‘C’	‘D’
LS0102	‘E’	‘F’
LS0103	‘G’	0

当要保存的数据是奇数字节时写入 “0”。

- 当寄存器地址长度为 32 位时

LS0100	‘A’	‘B’	‘C’	‘D’
LS0101	‘E’	‘F’	‘G’	0
LS0102				

当要保存的数据是奇数字节时写入 “0”。

- 模式 1

例如，当使用文件读取函数向寄存器地址中写入字符串 “ABCDEFG” 时

[w:[#INTERNAL]LS9130] = 1

_CF_read ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 7)

- 当寄存器地址长度为 16 位时

LS0100	‘B’	‘A’
LS0101	‘D’	‘C’
LS0102	‘F’	‘E’
LS0103	0	‘G’

当要保存的数据是奇数字节时写入 “0”。

- 当寄存器地址长度为 32 位时

LS0100	‘B’	‘A’	‘D’	‘C’
LS0101	‘F’	‘E’	0	‘G’
LS0102				

当要保存的数据是奇数字节时写入 “0”。

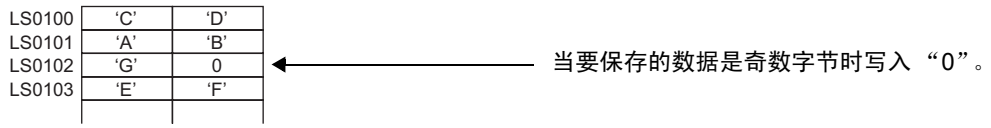
- 模式 2

例如，当使用文件读取函数向寄存器地址中写入字符串 “ABCDEFG”时

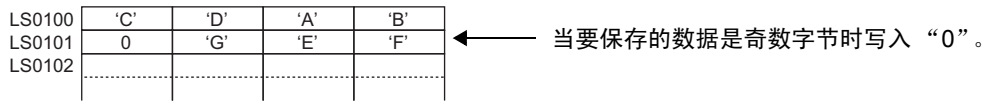
[w:[#INTERNAL]LS9130] = 2

_CF_read ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 7)

- 当寄存器地址长度为 16 位时



- 当寄存器地址长度为 32 位时



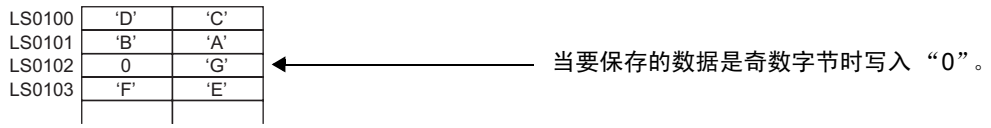
- 模式 3

例如，当使用文件读取函数向寄存器地址中写入字符串 “ABCDEFG”时

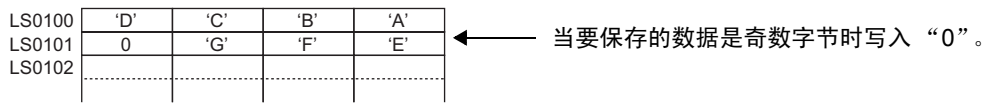
[w:[#INTERNAL]LS9130] = 3

_CF_read ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 7)

- 当寄存器地址长度为 16 位时



- 当寄存器地址长度为 32 位时



重要

- 数据存储模式与系统设置中的字符串数据模式不同。与字符串数据模式的关系如下表所示。

数据寄存器存储顺序	字字节 LH/HL 存储顺序	双字字节 LH/HL 存储顺序	D 脚本数据存储模式	文本数据模式
从起始数据保存	HL 顺序	HL 顺序	0	1
	LH 顺序		1	2
	HL 顺序	LH 顺序	2	5
	LH 顺序		3	4
从尾部数据开始保存	HL 顺序	HL 顺序		3
	LH 顺序			7
	HL 顺序	LH 顺序		8
	LH 顺序			6

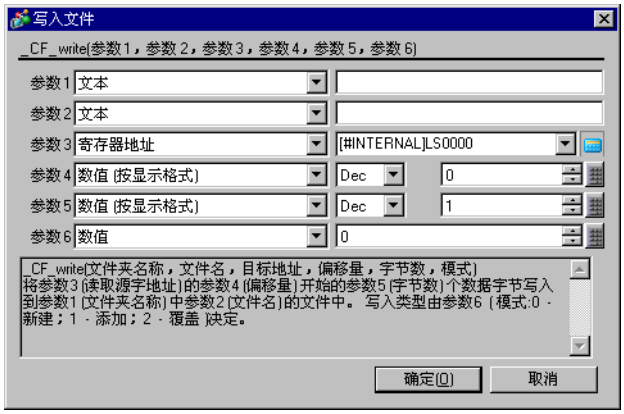
- 将数据重写至 CF 卡的频率有限制。因此，务必向另外一个存储介质定期备份所有 CF 卡数据。假定要重写 500KB 的 DOS 格式数据，限制次数是 10 万次。
- 如果在 CF 卡 /USB 存储器处理过程中发生错误，会将错误写至 CF 卡错误 / USB 存储器错误状态 [s:CF_ERR_STAT]/[s:USB_ERR_STAT] 中。更多信息，请参阅 " CF 卡 /USB 存储器错误状态 " (p21-103)。
- 不能将如下符号和字符用于文件夹名称或文件名。在文件夹名称或文件名中使用这些符号和字符将发生错误。

:	,	=	+	/	"	[
]	|	<	>	(空格)	?	

- 要指定根文件夹 (目录)，请将 “ ” (空字符串) 指定为文件夹名称。

■ 写入文件

项目	描述
摘要	可以选择如下三种模式的任意一种： “新建”、“添加”或 “覆盖”。有关数据存储顺序的更多信息，请参阅下面的 “数据存储模式”一节。
格式	_CF_write/_USB_write (文件夹名称、文件名、从地址中读取、偏移、字节数、模式)



参数 1
文件夹名称：固定字符串 (最大长度：32 个单字节字符)
参数 2
文件名： 固定字符串、内部寄存器 (最大长度：32 个单字节字符)、内部寄存器 + 临时地址
参数 3
读取源地址：寄存器地址、寄存器地址 + 临时地址
参数 4
偏移： 数值、寄存器地址、临时地址 (可以指定的最大数量：16 位长度为 65,535， 32 位长度为 4,294,967,295)
参数 5
字节数： 数值、寄存器地址、临时地址 (最大长度：1280)
参数 6
模式： 数值、寄存器地址、临时地址 (可用值：0, 1, 2)

存储格式概述

模式	名称	描述
0	新建	创建一个新文件。如果存在一个具有相同名称的文件，它将被删除。
1	添加	向指定文件添加数据。如果指定文件不存在，将会创建一个新文件。
2	覆盖	覆盖部分文件。如果指定偏移大于文件大小，将用 0 填充剩余区域，并将数据写入该区的后面。如果将偏移指定为文件数据的末尾，就相当于向文件添加数据。如果文件不存在，则会发生错误。有关该错误的更多信息，请参阅 " CF 卡 / USB 存储器错误状态 " (p21-103)。

表达式示例：

[w:[#INTERNAL]LS0200] = 0// 偏移 (当模式为 “新建” 时为 “0”)
[w:[#INTERNAL]LS0202] = 100 // 字节数 (100 个字节)
[w:[#INTERNAL]LS0204] = 0// 模式 (新建)
_CF_write ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100],
[w:[#INTERNAL]LS0200],
[w:[#INTERNAL]LS0202], [w:[#INTERNAL]:LS0204])

上例中，从 LS0100 中读取 100 字节的数据，以 DATA0001.BIN 为名保存在 \DATA 文件夹中。可以通过定义内部寄存器的偏移、字节数和模式间接定义字节数和模式。

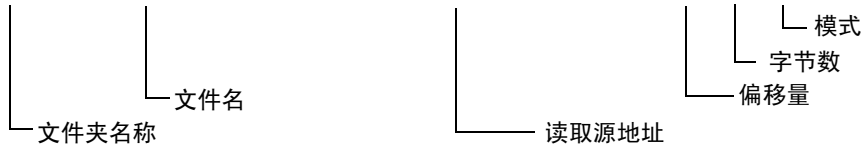
重要

- 偏移设置只在 “覆盖” 模式下有效。在 “新建” 和 “添加” 模式下禁用偏移设置。在非 “覆盖” 模式下将偏移值设置为 “0”。
- 当指定 “新建” 模式且已经存在一个具有相同文件名的文件时，它会被覆盖。
- 当为 “文件名” 指定 LS 区时，“读取源地址” 不被作为 D 脚本地址。
- 当为 “读取源地址” 指定 PLC 时，函数执行时仅从 PLC 中读取一次数据。如果数据读取过程中发生错误，它会导致 CF 卡或 USB 存储器读取错误：[s:CF_ERR_STAT] 或 [s:<USB_ERR_STAT]。当成功完成数据读取时该错误被清除。
- 数据被分成若干项目并从源地址中被读取，尽管这取决于要读取的字节数。因此，即便是数据读取过程中发生了通讯故障，仍可能已将数据部分地写入到了指定文件。
- 要为文件名指定完整路径，请将文件夹名称指定为 “*” (星号)。
  例如， _CF_read ("*", "\DATA\DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 10)

存储格式表达式示例

◆ **当指定 “新建” 模式时**

_CF_write ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 100, 0)



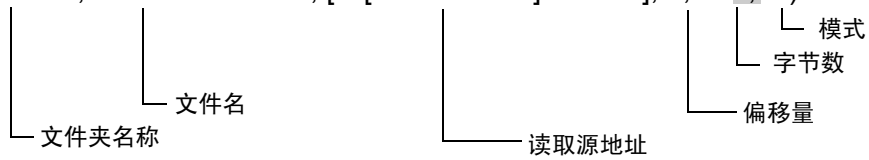
在上例中，从 LS0100 中读取 100 字节的数据， DATA0001.BIN 文件刚在 \DATA 文件夹中创建。

重 要

- 文件名只能使用 8.3 格式 (最大 12 个字符，其中 8 个字符用于文件名、点号， 3 个字符用于扩展名)。不能使用长度大于该格式的文件名。

◆ **当指定 “添加” 模式时**

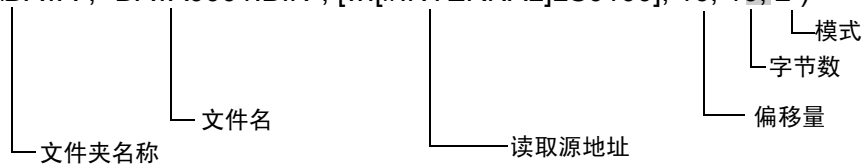
_CF_write ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 100, 1)



如果指定文件 (如例子中的 DATA0001.BIN) 已经存在且执行了上面的语句，会从 LS0100 和之后区域中读取 100 个字节的数据并将其写入 \DATA 文件夹中的 DATA0001.BIN 文件。

◆ **当指定 “覆盖” 模式时 (1)**

_CF_write ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 16, 10, 2)

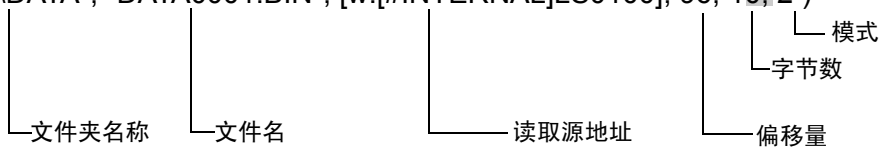


如果指定文件 (如例子中的 DATA0001.BIN) 已经存在且执行了上面的语句，就会读取保存在 LS0100 和之后区域的 10 个字节的数据，并覆盖保存在 \DATA 文件夹下 DATA0001.BIN 文件中的偏移之后第 17 个及之后的 10 个字节的数据。

◆ **当指定 “覆盖” 模式时 (2)**

(将被覆盖的文件小于偏移值和增加字节数总和。)

_CF_write ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 96, 10, 2)

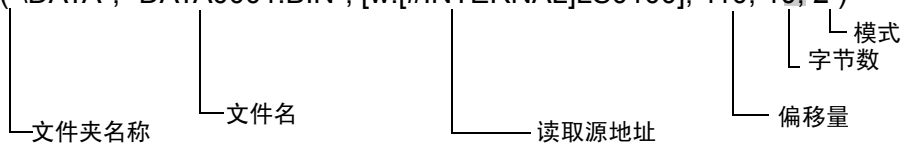


指定文件 (如例子中的 DATA0001.BIN) 已经存在且文件大小是 100 字节。如果为覆盖操作将偏移设置为 96 字节，字节数设置为 10 字节，会读取保存在 LS0100 及之后区域中的 10 字节数据。然后，前 4 个字节的已读取数据会覆盖保存在文件中的第 97 个及之后的 4 字节数据，剩余的 6 字节数据被添加到文件数据的末尾。最终文件就包含了 106 字节的数据。

◆ **当指定 “覆盖” 模式时 (3)**

(要覆盖的文件小于偏移值。)

_CF_write ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 110, 10, 2)



指定文件 (如例子中的 DATA0001.BIN) 已经存在且文件大小是 100 字节。如果为覆盖操作将偏移设置为 110 字节，字节数设置为 10 字节，则会用 0 填充第 101 字节和第 110 字节之间的区域，从 LS0100 及之后区域中读取的 10 字节数据会被写入第 111 字节及之后的字节中。最终文件就包含了 120 字节的数据。

重 要

- 第一个参数 (文件夹名称) 和第二个参数 (文件名) 的最大允许字符数是 32 个单字节字符。
- 可以为第二个参数 (文件名) 指定内部寄存器。指定内部寄存器可允许文件名间接地址指定。此外，最多可以使用 32 个单字节字符来指定文件名。
  例如， _CF_write ("\DATA", [w:[#INTERNAL]LS0100], [w:[#INTERNAL]LS0200], 0, 100, 0)
  在 LS0100 中保存文件名可允许文件名的间接地址指定。在本例中，按如下方式将文件名保存在 LS0100 至 LS0106 中。

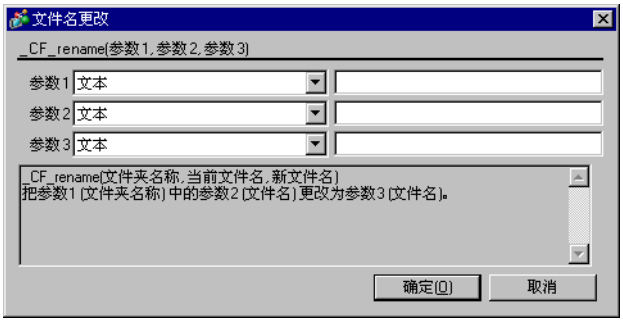
16 位

LS0100	‘D’	‘A’
LS0101	‘T’	‘A’
LS0102	‘0’	‘0’
LS0103	‘0’	‘1’
LS0104	‘.’	‘B’
LS0105	‘I’	‘N’
LS0106	‘\0’	‘\0’
	:	

← 文件名的末尾必须是 NULL 字符。人机界面将 NULL 字符前的数据作为文件名。

  在上例中，从 LS0200 中读取了 100 字节的数据并创建了一个用于保存数据的新文件 “\DATA\DATA0001.BIN”。
- 至于文件名，只能使用 “8.3 格式” (即最大 12 个字符，其中 8 个字符用于文件名、点号， 3 个字符用于扩展名)。不能使用长文件名。

■ 文件名更改

项目	描述
摘要	修改文件名。参数 1 指定 CF 卡数据文件夹。参数 2 指定原始文件名。参数 3 指定新名称。
格式	_CF_rename/_USB_rename (文件夹名称、文件名、新文件名) 也可以用 LS 地址间接指定文件名。  参数 1 文件夹名称：固定文本 参数 2 文件名：固定文本、内部寄存器、内部寄存器 + 临时地址 参数 3 文件名：固定文本、内部寄存器、内部寄存器 + 临时地址

表达式示例：

_CF_rename ("\DATA","DATA0001.BIN","DATA1234.BIN")

在上面的例子中，将文件名从 “\DATA\DATA0001.BIN” 改为 “\DATA\DATA1234.BIN”。

重 要

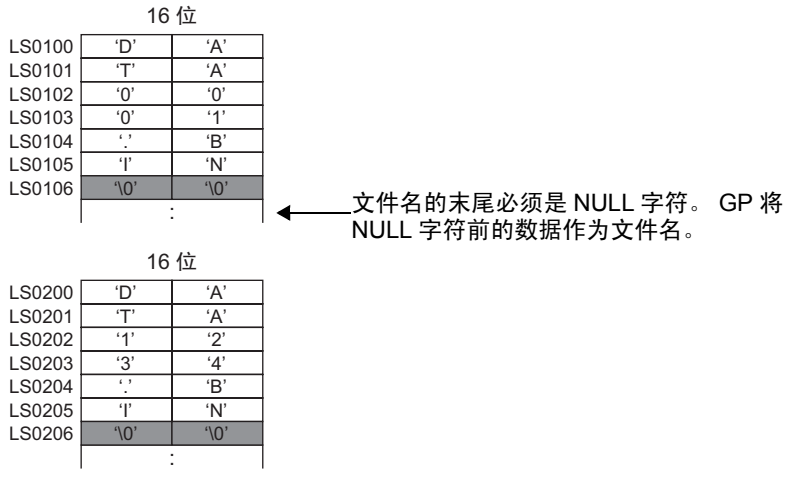
- 至于文件名，只能使用 “8.3 格式” (即最大 12 个字符，其中 8 个字符用于文件名、点号， 3 个字符用于扩展名)。不能使用长文件名。
- 第一个参数 (文件夹名称) 和第二个参数 (文件名) 的最大允许字符数是 32 个单字节字符。
- 可以为第二个和第三个参数 (文件名) 指定内部寄存器。指定内部寄存器可允许文件名间接地址指定。此外，最多可以使用 32 个单字节字符来指定文件名。

  示例

  _CF_rename ("\DATA", [w:[#INTERNAL]LS0100], [w:[#INTERNAL]LS0200])

  在 LS0100 和 LS0200 中保存文件名可支持文件名的间接地址指定。

- 按如下方式在 LS0100 至 LS0106 中保存文件名：

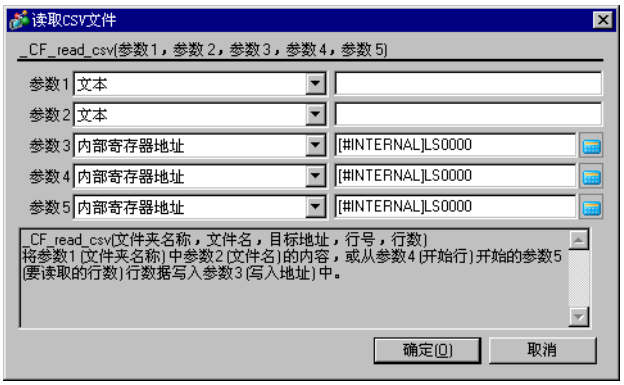


在上面的语句中，“\DATA\DATA0001.BIN” 文件被重命名为 “\DATA\DATA1234.BIN”。

- 当为 “文件名” 指定 LS 区时， LS 区不作为 D 脚本地址。
- 要指定根文件夹 (目录)，请将 “ ” (空字符串) 指定为文件夹名称。
- 要为文件名指定完整路径，请将文件夹名称指定为 “*” (星号)。

■ 读取 CSV 文件

项目	描述
摘要	从 CSV 文件 (用 “,” 分隔单元格内容而建立) 中以单元格为单位读取数据并将其写入字地址。
格式	_CF_read_csv/_USB_read_csv (文件夹名称、文件名、目标地址、起始行、行数)



参数 1： 文本 (最多 32 个单字节字符)
参数 2： 文本 (最多 32 个单字节字符)、内部寄存器、内部寄存器 + 临时地址
参数 3： 内部寄存器、用偏移量指定的内部寄存器
参数 4： 数值 (1 至 65,535)、内部寄存器、临时变量
参数 5： 数值 (1 至 65,535)、内部寄存器、临时变量

表达式示例：

_CF_read_csv ("\CSV", "SAMPLE.CSV", [w:[#INTERNAL]LS1000], 1, 2)

(当读取两行数据时，使用 “_CF_read_csv ( )” 函数从 CF 存储卡中的 [\CSV\SAMPLE.CSV] 文件的第一行开始。)

SAMPLE.CSV

```
001, "DAT01-01", "DAT01-2"
002, "DAT02-01", "DAT02-2"
```

读取两行数据，从 CSV 文件的第一行开始。如果第一个字符是数值 (“0” 至 “9” 或 “–”)，会将数据作为数值保存。如果第一个字符是 ["]，会将数据作为一个字符处理，并将 “00h” 保存在文本字符串的末尾。例如，当保存 “DAT01-01” 时，数据大小是 8 字符，这是一个偶数，总共将使用 5 个字：4 个字用于保存文本字符串，一个字用于在结尾保存 “00h”。例如，当保存 “DAT01-2” 时，数据大小是 7 字符，这是一个奇数，总共将使用 4 个字来保存该文本，在结尾保存 “00h”。

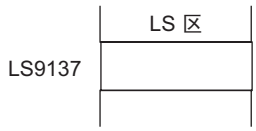
	16 位	
LS1000	1	
+1	‘D’	‘A’
+2	‘T’	‘0’
+3	‘1’	‘-’
+4	‘0’	‘1’
+5	00h	00h
+6	‘D’	‘A’
+7	‘T’	‘0’
+8	‘1’	‘-’
+9	‘2’	00h
LS1010	2	
+1	‘D’	‘A’
+2	‘T’	‘0’
+3	‘2’	‘-’
+4	‘0’	‘1’
+5	00h	00h
+6	‘D’	‘A’
+7	‘T’	‘0’
+8	‘2’	‘-’
+9	‘2’	00h

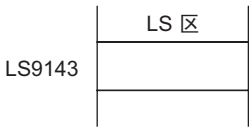
当“数据存储模式”为0时。

注 释

- 如果单元中的第一个字符是数值 (“0” 至 “9” 或 “-”)，会将值转换成数字数据，然后将数据写入 LS 寄存器。允许范围在 -32,768 至 32,767 之间。
- 如果单元格中的第一个字符是 ["]，会连同 ["] 一起将其作为文本字符串数据写入 LS 寄存器。当文本字符串数据的大小是奇数字节时，会在其末尾添加 “0x00”。当文本数据的大小是偶数字节时，会将 “0x0000” 写入紧跟最后一个地址的地址中。一个单元中最多可以输入 32 个单字节字符。
- 当 CSV 文件拥有两行以上数据时，可以从指定行开始读出所需行数。一行中最多可以输入 200 个单字节字符，一个 CSV 文件中最多可以输入 65535 个行。
- 当发生错误时，错误状态写入 LS9137 中 (对 USB 存储器来说是 LS9143)。
- 在将 CSV 文件文本数据写入 LS 寄存器时，数据存储顺序取决于数据存储模式。

错误状态



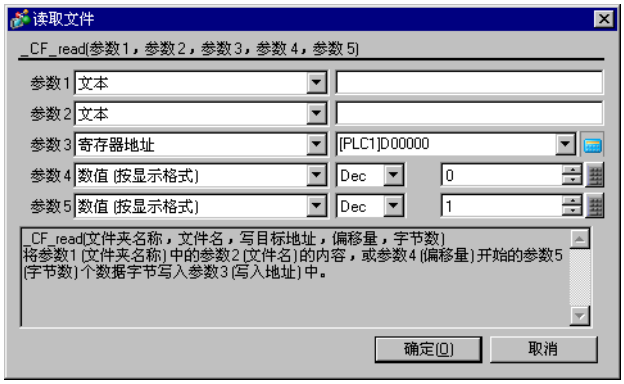


编辑器函数名称	LS 区	错误状态	原因
_CF_read_csv ( )/ _USB_read_csv ( )	LS9137/ LS9143	0000h	成功完成
		0001h	参数错误
		0002h	CF 卡 /USB 存储器错误 无 CF 卡或 USB 存储器 / 文件打开错误 / 文件读取错误
		0003h	写入 / 读取错误

重 要

- 如果将 “*” 指定为文件夹名，则可以为文件名指定完整路径。
- 文件名只能使用 8.3 格式 (最大 12 个字符，其中 8 个字符用于文件名、点号， 3 个字符用于扩展名)。不能使用长度大于该格式的文件名。
- 用于保存从 CSV 文件导入的数据的有效 LS 寄存器区限定为指定用户区 (LS20 至 LS2031 和 LS2096 至 LS8191)。
- 导入数据所需的处理时间与要读取的 CSV 文件的数据量成正例。在处理完成前部件不会更新。 (从总共 100 行 (每行 40 个字符) 的 CSV 文件的第 1 行一直到第 100 行中读取数据大概需要 10 秒钟的时间。)
- 和 "_CF_read()/_USB_read()" 函数不同，函数执行后不会立刻将状态保存到 [s:CF_ERR_STAT]/[s:USB_ERR_STAT]。 (在有些情况下，可能会保存一些未定义的值。)
- 请务必在以数字开始的文本字符串的开始和结束处插入 ["]。
  例如：
  [123, 2-D4EA] [123, "2-D4EA"]
  X O

■ 读取文件

项目	描述
摘要	在指定偏移后读取文件中指定字节数的数据并将它写入目标地址。有关数据存储顺序的更多信息，请参阅下面的 “数据存储模式”一节。
格式	_CF_read/_USB_read (文件夹名称、文件名、目标地址、偏移量、字节数)  参数 1 文件夹名称：固定字符串 (最大长度：32 个单字节字符) 参数 2 文件名： 固定字符串、内部寄存器、内部寄存器 + 临时地址 (最大长度：32 个单字节字符) 参数 3 写入地址：寄存器地址、寄存器地址 + 临时地址 参数 4 偏移： 数值、寄存器地址、临时地址 (可以指定的最大数量) 16 位长度为 65,535， 32 位长度为 4,294,967,295) 参数 5 字节数： 数值、寄存器地址、临时地址 (最大长度：1280)

表达式示例：

当偏移值为 16 时从指定文件中读取 16 字节数据：

_CF_read ("\DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 16, 16)

在上面的例子中，将 "\DATA\DATA0001.BIN" 文件中从第 17 个字节开始的 16 字节数据写入从 LS0100 开始的区。

重 要

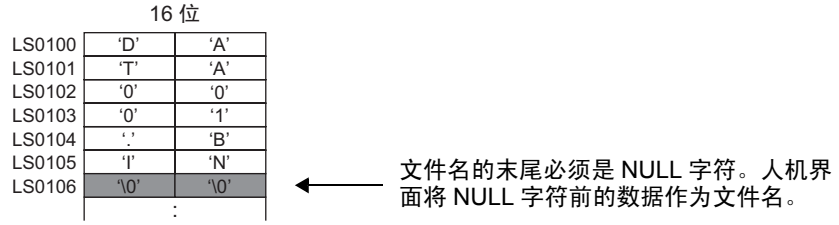
- 至于文件名，只能使用 “8.3 格式” (即最大 12 个字符，其中 8 个字符用于文件名、点号， 3 个字符用于扩展名)。不能使用长文件名。
- 第一个参数 (文件夹名称) 和第二个参数 (文件名) 的最大允许字符数是 32 个单字节字符。
- 可以为第二个参数 (文件名) 指定内部寄存器。指定内部寄存器可允许文件名间接地址指定。此外，最多可以使用 32 个单字节字符来指定文件名。

  示例

  当在 LS0100 及之后的区指定了文件且偏移值为 0 时读取文件中保存的 10 字节数据：

  _CF_read ("\DATA", [w:LS0100], [w:LS0200], 0, 10)

  在 LS0100 中保存文件名可允许文件名的间接地址指定。在本例中，按如下方式将文件名保存在 LS0100 至 LS0106 中。



  在上例中，读取 "\DATA\DATA0001.BIN" 文件开始处的 10 字节数据并将其写入 LS0200 及之后的区。

- 会将成功读取的字节数写入 CF 卡 /USB 存储器已读取字节 [s:CF_READ_NUM]/[s:USB_READ_NUM]。更多信息，请参阅 "21.11.5 CF 卡文件操作 /USB 文件操作 CF 卡 /USB 存储器错误状态 " (p21-103)。
- “文件名” 和 “写入地址” 中指定的内部寄存器不作为 D 脚本地址。
- 如果将 PLC 指定为写入地址，当字 (字节) 数增加时，向 PLC 写入数据需要更多时间。根据字数的多少，可能需要几秒钟的时间。
- 如果从文件中读取的数据超出了 PLC 的指定寄存器范围，就会发生通讯错误。在这种情况下，您必须关闭 PLC 的电源然后再次打开，将 PLC 从错误中复位。
- 如果将 PLC 寄存器指定为目标，由于 GP 至 PLC 的传输时间问题，不会立即写入数值。

  示例

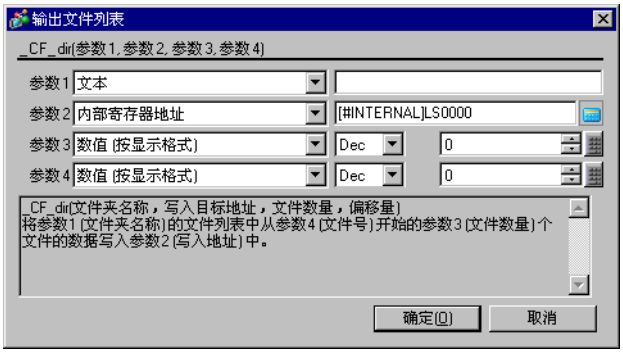
  在下面的脚本中，语句 (1) 从文件中读取 10 字节数据并将该数据写入 [w:D0100]。但是，由于传输时间的原因，在执行语句 (2) 时仍未将数据写入 [w:[PLC1]D0100] 。

```
_CF_read ("\DATA", "DATA0001.BIN", [w:[PLC1]D0100], 0, 10) .....(1)
[w:[PLC1]D0200] = [w:[PLC1]D0100] + 1 .....(2)
```

  在这种情况下，先在 LS 区中保存一次数据然后执行第二个语句，如下所示。

```
_CF_read ("\DATA", "DATA0001.BIN", [w:[PLC1]D0100], 0, 10)
memcpy ([w:[#INTERNAL]LS0100], [w:[PLC1]D0100], 10)
[w:[PLC1]D0200] = [w:[#INTERNAL]LS0100] + 1
```

■ 输出文件列表

项目	描述
摘要	指定文件夹中存在的文件列表被写入内部寄存器。参数 1 表示 CF 卡数据文件夹。参数 4 表示用来选择文件夹内的文件的偏移。参数 3 表示在该文件夹内选择的文件数。参数 2 指定文件将被写入的 LS 区。如果将偏移指定为 “0”，列表从第一个 (起始) 文件开始。
格式	_CF_dir/_USB_dir (文件夹名称、目标地址、文件数、偏移量)  参数 1 文件夹名称：固定文本 (最大长度：32 个单字节字符) 参数 2 写入地址：内部寄存器、用偏移量指定的内部寄存器 参数 3 文件数：数值、寄存器地址、临时地址 (最大长度：32) 参数 4 数值、寄存器地址、临时地址

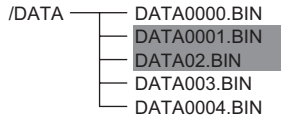
表达式示例：

当偏移值为 1 时 (第二个文件) 输出包含两个文件的文件列表：

_CF_dir ("\DATA*.*", [w:[#INTERNAL]LS0100], 2, 1)

当 DATA 文件夹中存在如下文件时执行了上述语句，会在 LS0100 及之后的区写入文件名 "DATA0001.BIN" 和 "DATA02.BIN" 。

(文件夹中的内容)



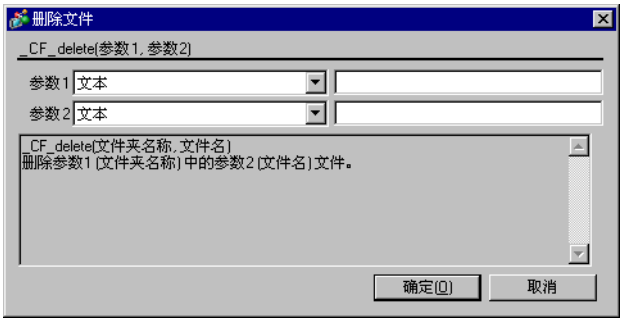
(LS区的内容)

	16 位		
LS0100	'D'	'A'	使用了7个字。
LS0101	'T'	'A'	
LS0102	'0'	'0'	
LS0103	'0'	'1'	
LS0104	'.'	'B'	
LS0105	'I'	'N'	
LS0106	'\0'	'\0'	
LS0107	'D'	'A'	使用了7个字。
LS0108	'T'	'A'	
LS0109	'0'	'2'	
LS0110	'.'	'B'	
LS0111	'I'	'N'	
LS0112	'\0'	'\0'	
LS0113	'\0'	'\0'	

重 要

- 如果将偏移指定为 “0”，列表从第一个 (起始) 文件开始。
- 至于文件名，只能使用 “8.3 格式” (即最大 12 个字符，其中 8 个字符用于文件名、点号， 3 个字符用于扩展名)。不能使用长文件名。
- 如果指定的文件夹中包含的文件数没有指定的那么多，将用 NULL 字符 (“\0”) 填充剩余 LS 区。
- 如果文件名小于 12 个字符，将用 NULL 字符 (“\0”) 填充空位。
- 指定文件夹名时，例如 “\DATA*.*”，一定要添加 “*.*”。星号 *.* 表示显示所有文件。
- 实际列出的文件数被写入 CF 卡 /USB 存储器列出文件 [s:CF_FILELIST_NUM]/[s:USB_FILELIST_NUM] 中。
  更多信息，请参阅 " CF 卡 /USB 存储器错误状态 " (p21-103)
- 写入 LS 地址不作为 D 脚本地址。
- 将文件名写入 LS 区时不对它们进行排序。按创建顺序 (FAT 输入顺序) 写入它们。
- 您可以通过指定文件扩展名来创建列表。要列出具有某一扩展名的文件，可使用 “\DATA*.BIN” 的格式。但是，不能在文件名中使用 "*"。

■ 删除文件

项目	描述
摘要	从 CF 卡中删除指定文件。参数 1 表示 CF 卡数据文件夹。参数 2 表示要删除的文件的名称。
格式	_CF_delete/_USB_delete (文件夹名称、文件名) 也可以用 LS 地址间接指定文件名。  参数 1 文件夹名称：固定文本 参数 2 文件名：固定文本、内部寄存器、内部寄存器 + 临时地址

表达式示例：

_CF_delete ("\DATA", "DATA0001.BIN")

上面的例子删除 “\DATA\DATA0001.BIN” 文件。

重 要

- 至于文件名，只能使用 “8.3 格式” (即最大 12 个字符，其中 8 个字符用于文件名、点号， 3 个字符用于扩展名)。不能使用长文件名。
- 第一个参数 (文件夹名称) 和第二个参数 (文件名) 的最大允许字符数是 32 个单字节字符。
- 可以为第二个参数 (文件名) 指定内部寄存器。指定内部寄存器可允许文件名间接地址指定。此外，最多可以使用 32 个单字节字符来指定文件名。

  在本例中，按如下方式将文件名保存在 LS0100 至 LS0106 中。

	16 位	
LS0100	‘D’	‘A’
LS0101	‘T’	‘A’
LS0102	‘0’	‘0’
LS0103	‘0’	‘1’
LS0104	‘.’	‘B’
LS0105	‘I’	‘N’
LS0106	‘\0’	‘\0’
	:	

← 文件名的末尾必须是 NULL 字符。人机界面将 NULL 字符前的数据作为文件名。

  在上例中，“\DATA\DATA0001.BIN” 文件被删除。
- 要指定根文件夹 (目录)，请将 “ ” (空字符串) 指定为文件夹名称。
- 当为 “文件名” 指定 LS 区时，“写入地址” 不作为 D 脚本地址。
- 要为文件名指定完整路径，请将文件夹名称指定为 “*” (星号)。

21.11.6 打印机操作

打印机操作	函数摘要
	标签设置 ☞" ■ 标签设置 " (p21-124) 从控制和状态变量中指定。
	发送 ☞" ■ 发送 " (p21-126) 向 COM 端口输出指定字节数。

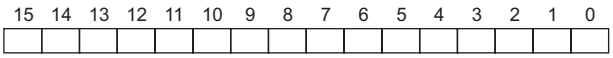
重 要
- COM1 或 USB/PIO (USB-PIO) 是可以用做打印机操作函数的端口。

■ 标签设置

控制

控制 (PRN_CTRL) 是用于清空发送缓冲器和错误状态的变量。该变量是只写的。

- 控制 (PRN_CTRL) 摘要



位	内容
15	保留
14	
13	
12	
11	
10	
9	
8	
7	
6	
5	
4	
3	
2	1: 清除错误
1	保留
0	1: 清楚发送缓冲器

重 要
- 如果选择字，且同时置位两个以上的位，按如下顺序执行处理：

  清除错误

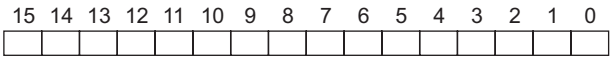
  ↓

  清空发送缓冲器
- 不要使用保留位。只设置需要的位。

状态

使用状态变量 (PRN_STAT) 的目的是为了检查发送缓冲器中数据的存在状态及获取错误状态。该状态变量为只写。

- 状态变量 (PRN_STAT) 的内容



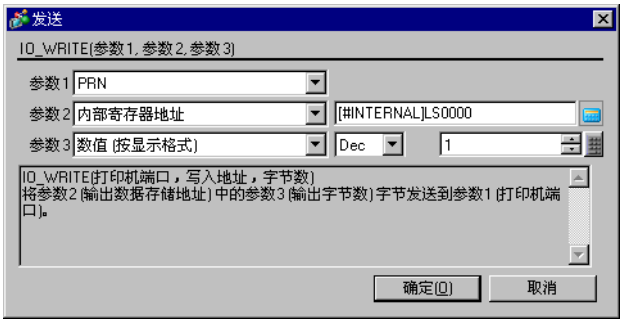
位	内容
15	保留
14	打印机接口ERROR信号的状态 打印机错误（输入）： 0: 错误 1: 正常
13	打印机接口SLCT信号的状态 选择（输入）： 0: 离线 1: 在线
12	打印机接口PE信号的状态 缺纸（输入）： 0: 正常 1: 缺纸
11	保留
10	
9	
8	
7	
6	
5	
4	
3	
2	
1	0: 正常 1: 发送错误
0	0: 数据在发送缓冲器中已经存在 1: 发送缓冲器为空

重 要

- 如果发送缓冲器溢出，就会发生错误。当发生该错误时，传输错误位置 ON。
- 发送缓冲器是 8,192 个字节。
- 可能会在将来为保留位分配数据。因此，请务必只检查必需的位。

■ 发送

项目	描述
摘要	向 COM 端口输出指定字节数。无论指定何种打印机类型，都会输出数据。
格式	IO_WRITE ([p:PRN]，输出数据存储地址、输出字节数)



参数 1：　[p:PRN]
参数 2：　内部寄存器
参数 3：　整数值、寄存器地址、临时地址

重 要

- 可以为参数 3 指定的最大值是 1024。即便指定了大于 1024 的值时，也只能从 COM 串口中输出 1024 字节数据。

表达式示例 1：

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 10)

在上例中，从 COM 端口输出保存在 LS1000 及之后区中的 10 字节数据。

表达式示例 2：

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS0800])

在上例中，从 COM 端口中输出保存在 LS1000 及之后区中的数据。字节数同写入 LS0800 中的字节数一样。

表达式示例 3：

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS 1000], [t:0010])

在上例中，从 COM 端口中输出保存在 LS1000 及之后区中的数据。字节数同写入临时地址 [t:0010] 中的字节数一样。

数据存储模式

如果在执行 COM 端口操作函数时从寄存器地址中读取数据，您可以指定读出数据的存储顺序。

在 LS9130 中设置数据存储模式可以更改存储顺序。

可以从如下四种模式中任意选择一种：0, 1, 2 或 3。

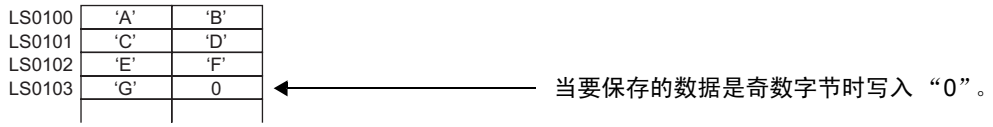
◆ 模式 0

例如，当用 COM 端口操作函数从寄存器地址中读取字符串 “ABCDEFG”时

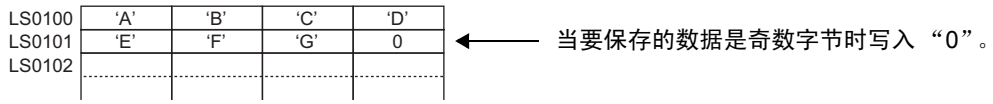
[w:[#INTERNAL]LS9130] = 0

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 7)

- 当寄存器地址长度为 16 位时



- 当寄存器地址长度为 32 位时



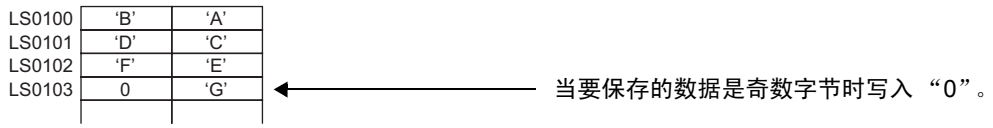
◆ 模式 1

例如，当用 COM 端口操作函数从寄存器地址中读取字符串 “ABCDEFG”时

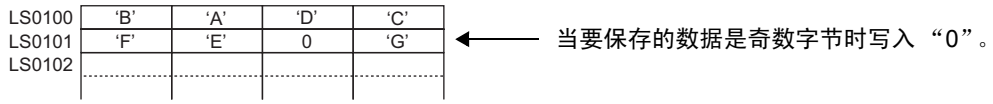
[w:[#INTERNAL]LS9130] = 1

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 7)

- 当寄存器地址长度为 16 位时



- 当寄存器地址长度为 32 位时



◆ 模式 2

例如，当用 COM 端口操作函数从寄存器地址中读取字符串 “ABCDEFG”时

[w:[#INTERNAL]LS9130] = 2

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 7)

- 当寄存器地址长度为 16 位时

LS0100	‘C’	‘D’
LS0101	‘A’	‘B’
LS0102	‘G’	0
LS0103	‘E’	‘F’

← 当要保存的数据是奇数字节时写入 “0”。

- 当寄存器地址长度为 32 位时

LS0100	‘C’	‘D’	‘A’	‘B’
LS0101	0	‘G’	‘E’	‘F’
LS0102				

← 当要保存的数据是奇数字节时写入 “0”。

◆ 模式 3

例如，当用 COM 端口操作函数从寄存器地址中读取字符串 “ABCDEFG”时

[w:[#INTERNAL]LS9130] = 3

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 7)

- 当寄存器地址长度为 16 位时

LS0100	‘D’	‘C’
LS0101	‘B’	‘A’
LS0102	0	‘G’
LS0103	‘F’	‘E’

← 当要保存的数据是奇数字节时写入 “0”。

- 当寄存器地址长度为 32 位时

LS0100	‘D’	‘C’	‘B’	‘A’
LS0101	0	‘G’	‘F’	‘E’
LS0102				

← 当要保存的数据是奇数字节时写入 “0”。

重 要

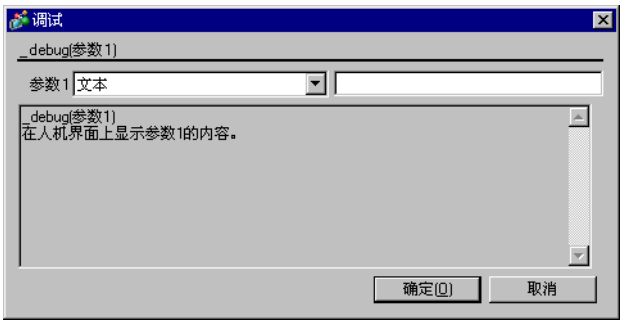
- 数据存储模式与系统设置中的字符串数据模式不同。与字符串数据模式的关系如下表所示。

数据寄存器存储顺序	字字节 LH/HL 存储顺序	双字字节 LH/HL 存储顺序	D 脚本数据存储模式	文本数据模式
从起始数据保存	HL 顺序	HL 顺序	0	1
	LH 顺序		1	2
	HL 顺序	LH 顺序	2	5
	LH 顺序		3	4
从尾部数据开始保存	HL 顺序	HL 顺序		3
	LH 顺序			7
	HL 顺序	LH 顺序		8
	LH 顺序			6

21.11.7 其他

其他	函数摘要
	调试函数 ☞" ■ 调试函数 " (p21-129) 在画面上显示指定地址或文本以便进行调试。
	启动应用程序 ☞" ■ 启动应用程序 " (p21-131) 运行指定范围并启动应用程序。
	WinGP，退出 ☞" ■ 退出 WinGP" (p21-133) 退出 WinGP。

■ 调试函数

项目	描述
摘要	在画面上显示指定地址或文本以便进行调试。 在完成调试后，清除脚本编辑器的 [启用调试函数] 复选框，不会删除任何脚本。只有调试画面不会出现。
格式	_debug (参数 1)  参数 1：　　文本 (最多 32 个单字节、 16 个双字节字符)

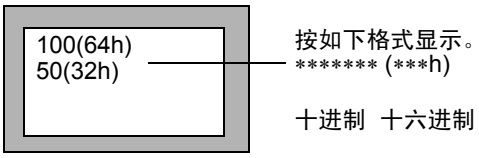
参数 1 的内容

参数 1	格式	描述
文本	_debug ("ABC")	显示 “ ” 内的文本。文本最多可以有 32 个单字节字符。
字地址或临时地址	_debug (w:[PLC1]D1000)	显示已设置的字地址或临时地址值。
换行	_debug (_CRLF)	将光标移到下一行开始的位置。
归位	_debug (_CR)	将光标移到同一行开始的位置。

◆ 表达式示例 1：

以下脚本显示字地址值。

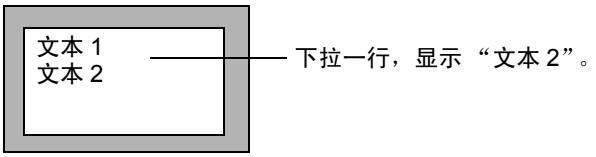
```
[w:[#INTERNAL]LS0100]=100
_debug
([w:[#INTERNAL]LS0100])
_debug (_CRLF)
[w:[#INTERNAL]LS0100]=50
_debug
([w:[#INTERNAL]LS0100])
```



◆ 表达式示例 2：

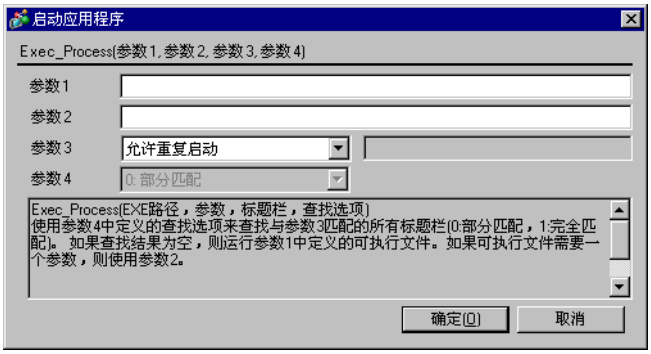
以下脚本显示换行和文本。

```
_debug ("Test1")
_debug (_CRLF)
_debug ("Test2")
```



■ 启动应用程序

该功能在非 IPC 系列的型号上不起作用。

项目	描述
摘要	运行指定范围并启动应用程序。 可以指定一些设置，如启动参数和禁止多重启动。
格式	Exec Process (参数 1, 参数 2, 参数 3, 参数 4)  参数 1 EXE 路径：输入您想启动的应用程序的可执行文件 (.exe) 的绝对路径。最多可以输入 255 个字符。 参数 2 参数：输入可执行文件的启动参数。最多可以输入 255 个字符。 参数 3 标题栏： 如果不允许多重启动，选择 “禁止多重启动”并在 [标题栏] 中输入标题。最多可以输入 63 个字符。 如果发现了另外一个标题与 [窗口标题] 相同的窗口，应用程序就不能启动。如果选择了 [允许多重启动] 或未指定 [窗口标题]，就允许多重启动。 参数 4 仅查找完全匹配的标题： 只有当您选择了参数 3 “不允许重复启用”时才可用。当选择了 “0：部分匹配”时，如果发现了一个与 [标题栏] 的标题部分相同的窗口，就不会执行指定应用程序。当选择了 “1：完全匹配”时，如果发现了一个与 [标题栏] 的标题完全相同的窗口，就不会执行指定应用程序。

注 释

- 参数 1 要求文本 (EXE 路径)。如果您未输入文本，会发生错误。
- 该功能在非 IPC 系列的型号上不起作用。

参数 1(EXE 路径) 输入方法

有三种输入 EXE 路径的方法：

如下描述给出了一个在 C:\Documents and Settingsuser\Local Settings\Temp 路径中执行 sample.exe 的示例。

1. 完整路径指定
   例如，C:\Documents and Settings\user\Local Settings\Temp\sample.exe
2. 仅 EXE 名称
   如果可执行文件位于在 IPC 系列的环境设置中作为路径指定的文件夹中。
   例如， sample.exe
   (如果设置是 Path=C:\Documents and Settings\user\Local Settings\Temp 时启动)
3. 用环境变量来定义路径
   如果可执行文件位于 IPC 系列上环境设置中的环境参数指定的文件夹中。
   例如， %TEMP%\sample.exe
   (如果将环境参数指定为 TEMP=C:\Documents and Settings\user\Local Settings\Temp 时启动)

表达式示例 1：

允许多重启动 (启动记事本并显示 Readme.txt)

```
Exec_Process
("C:\WINDOWS\SYSTEM32\notepad.exe","D:\TEMP\Readme.txt","",0)
Exec_Process ("%SystemFolder%\notepad.exe","D:\TEMP\Readme.txt","",1)
```

表达式示例 2：

禁止多重启动：

部分匹配 (启动记事本并显示 Readme.txt)

```
Exec_Process
("C:\WINDOWS\SYSTEM32\notepad.exe","D:\TEMP\Readme.txt","Readme",0)
```

表达式示例 3：

禁止多重启动：仅全字匹配 (启动记事本并显示 Readme.txt)

```
Exec_Process
("C:\WINDOWS\SYSTEM32\notepad.exe","D:\TEMP\Readme.txt","Readme.txt -
Notepad",1)
```

表达式示例 4：

禁止多重启动：部分匹配 (启动记事本)

```
Exec_Process ("C:\WINDOWS\SYSTEM32\notepad.exe","","Notepad",0)
```

表达式示例 5：

无参数 (启动记事本)

```
Exec_Process ("C:\WINDOWS\SYSTEM32\notepad.exe","","",0)
```

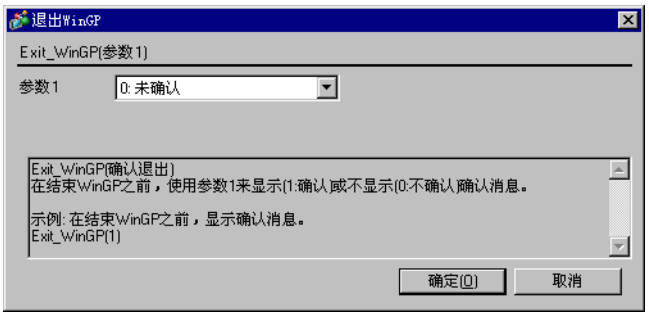
表达式示例 6：

多个参数 (启动 sample.exe)

```
Exec_Process ("C:\WINDOWS\SYSTEM32\sample.exe","/v /a/s","",1)
```

■ 退出 WinGP

该功能在非 IPC 系列的型号上不起作用。

项目	描述
摘要	退出 WinGP。您可以在退出时显示一条确认消息。
格式	Exit_WinGP(参数 1)  参数 1 文件夹名称：选择 “ 0 ： 不确认” 或 “1： 确认”。

注 释

- 参数 1 要求文本 (EXE 路径)。如果您未输入文本，会发生错误。
- 当您将 “退出 WinGP” 脚本移植到非 IPC 系列机型上时该功能不起作用。

表达式示例：

当退出 WinGP 时显示一条确认消息。

Exit_WinGP(1)

21.11.8 条件表达式

条件表达式	函数摘要
描述表达式 if - endif if - else - endif loop - endloop break return	if - endif ☞" ■ if-endif" (p21-134) 当括号 “( )” 内的 “if” 条件成立时，就运行 “if( )” 语句后的表达式。
	if - else - endif ☞" ■ if-else-endif" (p21-134) 当用括号 “( )” 括起来的 “if” 条件成立时，就运行 “if ( )” 语句后的表达式。当条件不成立时，就运行 “else” 表达式。
	loop - endloop ☞" ■ loop - endloop" (p21-135) 根据 “loop” 后的括号 "( )" 中指定的临时地址保存的数重复循环处理。
	break ☞" ■ break" (p21-137) 在执行 loop ( ) 循环过程中停止循环操作。
	return ☞" ■ return" (p21-137) 从开始再执行一次。 只能在扩展脚本中使用它。

■ if-endif

当括号 “( )” 内的 “if” 条件成立时，就运行 “if( )” 语句后的表达式。

注 释
- 条件表达式中不能使用赋值 "=" 字符。

■ if-else-endif

当用括号 “( )” 括起来的 “if” 条件成立时，就运行 “if ( )” 语句后的表达式。当条件不成立时，就运行 “else” 表达式。

注 释
- 条件表达式中不能使用赋值 "=" 字符。

■ loop - endloop

根据 “loop” 后的括号 "( )" 中指定的临时地址保存的数重复循环处理。

无限循环

当 loop 括号 ( ) 中没有语句时循环是无限的。
您可以在扩展脚本中使用无限循环。

表达式示例：

```
loop ( )
{
    [w:[#INTERNAL]LS0100]=[w:[#INTERNAL]LS0100]+1
    if ( [w:[#INTERNAL]LS0100] >10)
    {
        break
    }
    endif
}
endloop
```

注 释

- loop ( ) 的格式如下：
  例如：

```
loop( 循环次数 )// 定义保存循环次数的临时地址。
    {
        操作等式
        break    // 用于中途退出循环体 ( 可选 )
    } endloop   // 定义循环的终止
```

- 只能在圆括号中输入一个临时字地址。 (例如， loop ([t:000]))
- “loop ( )” 不能用于触发等式。
- 用于定义循环次数的临时字地址值在每次循环时都减少。当该值为 0 时，循环操作终止。如果修改了为循环次数定义的临时字地址值，该循环可能变成无限循环。这里使用的临时字地址被指定为全局地址。因此，为其他目的同时使用该临时字地址可能导致无限循环。
- 在循环操作结束前，将不会更新或刷新部件等的画面显示。
- loop ( ) 可以嵌套。如果循环含有嵌套，使用 “break” 命令可以跳过最内层的 loop ( )。

```
loop ([t:0000]) // loop 1
{
    loop  ([t:0001]) // loop 2
    {
        break    // 从 loop 2 中退出
    }endloop

    break        // 从 loop 1 中退出
}endloop
```

- 如果循环操作在没有使用退出命令的情况下完成，临时字地址值就变成 0。

注 释

- 临时字地址值的可用范围根据所使用的数据格式 (Bin, BCD)、位长和代码 +/- 而有所不同。如果已经设置了代码 +/- 且临时字地址值变成了负值，将在循环开始时判断该条件，且循环处理停止。
- 在循环中请不要使用 PLC 寄存器。而应使用人机界面内部 LS 区用户区地址或临时字地址。例如，以下描述在短时间内多次向 PLC 执行数据写入操作 (在下例中是 100 次)。由于不能以这样的速度执行通讯处理 (写入 PLC 所需的时间)，因此这会导致系统错误。

  例如：

```
[t:0000] = 100                                          // 循环 100 次
loop ([t:0000])
{
  [w:[PLC1]D0200] = [w:[#INTERNAL]LS0100]               // 写入 D0200
  [w:[#INTERNAL]LS0100] =
  [w:[#INTERNAL]LS0100] + 1                             // LS0100 加 1
}endloop
```

  请做如下修改：

```
[t:0000] = 100                                          // 循环 100 次
loop ([t:0000])
{
  [w:[#INTERNAL]LS0200] =
  [w:[#INTERNAL]LS0100]                                 // 写入 D0200
  [w:[#INTERNAL]LS0100] =
  [w:[#INTERNAL]LS0100] + 1                             // LS0100 加 1
}endloop
[w:[PLC1]D0200]=[w:[#INTERNAL]LS0200]                   //LS0200 内容，写入 D0200
```

- 将 “loop” 或 “break” 用作 D 脚本函数的函数名会造成错误。

■ break

在 loop ( ) 运算过程中退出 loop 运算。

注 释

- "break" 命令只能在 loop ( ) 的 { } 节中使用。
- 如果在 if {} 表达式中使用 “break” 命令，脚本就不能正确运行。

■ return

如果 “自定义函数” 包含 "return"，该函数的处理终止，控制返回到函数的调用程序。

如果主函数包含 “return”
立即取消主函数的处理并从主函数开始的地方重新开始。

注 释

- 条件表达式中不能使用赋值 "=" 字符。

表达式示例：

```
[w:[#INTERNAL]LS0100]=([w:[#INTERNAL]LS0200]>> 8) & 0xFF
if ([w:[#INTERNAL]LS0100]==0)        // 当 LS0100 为 "0" 时，不再执行处理
{
    set([b:[#INTERNAL]LS005000])     // 为错误显示设置位地址
    return                           // 结束
}
endif
```

21.11.9 比较

比较	函数摘要
比较 逻辑与 (AND) 逻辑或 (OR) 逻辑非 (NOT) 小于 (<) 小于等于 (<=) 不等于 (<>) 大于 (>) 大于等于 (>=) 等于 (==)	逻辑与 (AND) ☞" ■ 逻辑与 (AND)" (p21-138) N1 and N2: 如果 N1 和 N2 同时为 ON 时成立。
	逻辑或 (OR) ☞" ■ 逻辑或 (OR)" (p21-138) N1 or N2: 如果 N1 和 N2 中有一个为 ON 时成立。
	非 (NOT) ☞" ■ 非 (NOT)" (p21-138) not N1: 如果 N1 是 1 时为 0，如果 N1 是 0 时为 1。
	小于 (<) ☞" ■ 小于 (<)" (p21-139) 如果 N1 大于 N2(N1<N2) 时成立。
	小于或等于 (=) ☞" ■ 小于等于 (<=)" (p21-139) N1 小于等于 N2(N1<=N2) 时成立。
	不等于 (<>) ☞" ■ 不等于 (<>)" (p21-139) 如果 N1 不等于 N2(N1<>N2) 时成立。
	小于 (>) ☞" ■ 大于 (>)" (p21-139) 如果 N1 大于 N2(N1>N2) 时成立。
	大于等于 (>=) ☞" ■ 大于等于 (>=)" (p21-139) 如果 N1 大于等于 N2(N1>=N2) 时成立。
	等于 (==) ☞" ■ 等于 (==)" (p21-139) N1 等于 N2(N1=N2) 时成立。

■ 逻辑与 (AND)

AND 右侧和左侧。值为 0(零) 表示 OFF，其他值表示 ON。
N1 and N2: 如果 N1 和 N2 同时为 ON 时成立。否则为假。

■ 逻辑或 (OR)

OR 右侧和左侧。值为 0(零) 表示 OFF，其他值表示 ON。
N1 or N2: 如果 N1 和 N2 中有一个为 ON 时成立。否则为假。

■ 非 (NOT)

反转该值。 0(零) 被认为是 1，其他值则被认为是 0。
not N1: 如果 N1 是 1 时为 0，如果 N1 是 0 时为 1。

■ 小于 (<)

比较两个字地址中的数据，或比较一个字地址中的数据和一个常量。
如果 N1 大于 N2(N1<N2) 时成立。

■ 小于等于 (<=)

比较两个字地址中的数据，或比较一个字地址中的数据和一个常量。
N1 小于等于 N2(N1<=N2) 时成立。

■ 不等于 (<>)

比较两个字地址中的数据，或比较一个字地址中的数据和一个常量。如果 N1 不等于 N2(N1<>N2) 时成立。

■ 大于 (>)

比较两个字地址中的数据，或比较一个字地址中的数据和一个常量。
如果 N1 大于 N2(N1>N2) 时成立。

■ 大于等于 (>=)

比较两个字地址中的数据，或比较一个字地址中的数据和一个常量。
如果 N1 大于等于 N2(N1>=N2) 时成立。

■ 等于 (==)

比较两个字地址中的数据，或比较一个字地址中的数据和一个常量。
N1 等于 N2(N1=N2) 时成立。

命令		例如
逻辑与	and	if ((运算) and (运算))
逻辑或	or	if ((运算) or (运算))
非	not	if (not (运算))
小于	<	(项 1) < (项 2)
小于等于	<=	(项 1) <= (项 2)
不等于	<>	(项 1) <> (项 2)
大于	>	(项 1) > (项 2)
大于等于	>=	(项 1) >= (项 2)
等于	==	(项 1) == (项 2)

21.11.10 运算符

运算符	函数摘要
运算符 加[+] 减[-] 取模[%] 乘[*] 除[/] 赋值[=] 左移[<<] 右移[>>] 按位与[&] 按位或[|] 按位异或[^] 按位取反[~]	加 (+) ☞" ■ 加 (+)" (p21-141) 将两个字地址中的数据相加，或将一个字地址中的数据和一个常量相加。
	减 (-) ☞" ■ 减 (-)" (p21-141) 将两个字地址中的数据相减，或将一个字地址中的数据和一个常量相减。
	取模 (%) ☞" ■ 取模 (%)" (p21-141) 检测两个字地址中的数据执行除法运算后的余数，或一个字地址中的数据和常量执行除法运算的余数。
	乘 (*) ☞" ■ 乘 (*)" (p21-141) 将两个字地址中的数据相乘，或将一个字地址中的数据和一个常量相乘。
	除 (/) ☞" ■ 除 (/)" (p21-141) 将两个字地址中的数据或一个字地址中的数据用一个常量去除。
	赋值 (=) ☞" ■ 赋值 (=)" (p21-141) 把右手侧的值赋给左手侧。
	左移 (<<) ☞" ■ 左移 (<<)" (p21-141) 按右侧数字将左侧数据向左移。
	右移 (>>) ☞" ■ 右移 (>>)" (p21-142) 按右侧数字将左侧数据向右移。
	按位与 (&) ☞" ■ 按位与 (&)" (p21-142) 执行字地址数据间的逻辑与，或字地址数据和常量间的逻辑与。
	按位或 (|) ☞" ■ 按位或 (|)" (p21-142) 执行字地址数据间的逻辑或，或字地址数据和常量间的逻辑或。

运算符	函数摘要
运算符	按位异或 (^) ☞" ■ 按位异或 (^)" (p21-142) 执行字地址数据间的异或，或字地址数据和常量间的异或。
	按位取反 (~) ☞" ■ 按位取反 (~)" (p21-142) 反转位。

■ 加 (+)

将两个字地址中的数据相加，或将一个字地址中的数据和一个常量相加。当计算结果溢出时，数字被截取。

■ 减 (-)

将两个字地址中的数据相减，或将一个字地址中的数据和一个常量相减。当计算结果溢出时，数字被截取。

■ 取模 (%)

检测两个字地址中的数据执行除法运算后的余数，或一个字地址中的数据和常量执行除法运算的余数。运算结果可能取决于左侧和右侧的符号。

■ 乘 (*)

将两个字地址中的数据相乘，或将一个字地址中的数据和一个常量相乘。当计算结果溢出时，数字被截取。

■ 除 (/)

将两个字地址中的数据或一个字地址中的数据用一个常量去除。运算中得到的小数值将被省去。当计算结果溢出时，数字会被截取。

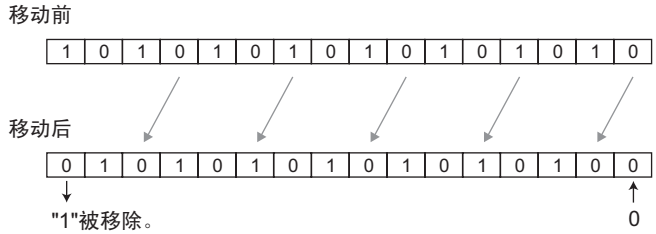
■ 赋值 (=)

将右侧的值赋给左侧。只能指定左侧的地址。右侧可以是地址和常量。当计算结果溢出时，数字会被截取。

■ 左移 (<<)

按右侧数字将左侧数据向左移。该功能仅支持逻辑移位。

例如，左移操作 (左移一位)。



■ 右移 (>>)

右移左侧的数据，移动的位数为右侧的数字。该功能仅支持逻辑移位。

■ 按位与 (&)

执行字地址数据间的逻辑与，或字地址数据和常量间的逻辑与。用来提取特定位或屏蔽位的特定字串。

■ 按位或 (|)

执行字地址数据间的逻辑或，或字地址数据和常量间的逻辑或。用于将特定位置 ON。

■ 按位异或 (^)

执行字地址数据间的异或，或字地址数据和常量间的异或。

■ 按位取反 (~)

反转位。

注 释

- 有关截除运算结果中的小数或溢出位的更多信息，请参阅
☞"21.10.4 运算结果注释 " (p21-63)

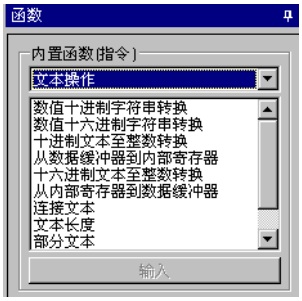
优先级和相关性

下表显示了运算符的优先级。如果两个以上运算符具有相同的优先级，按照相关性显示的方向进行运算。

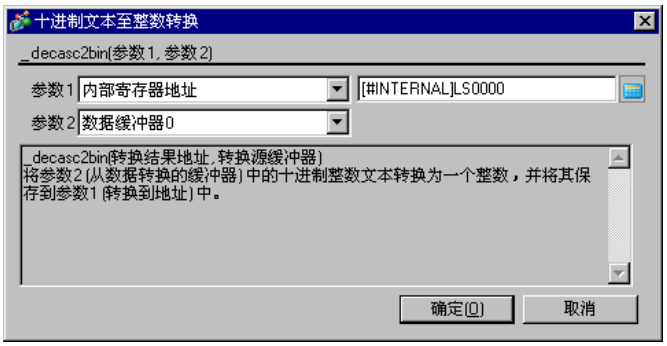
优先级	运算符	相关性
高	( )	->
	not ~	<-
	* / %	->
	+ -	->
	<< >>	->
	< <= > >=	->
	== <>	->
	& ^ |	->
	and or	->
低	=	<-

21.11.11 文本操作

只能在扩展脚本中使用文本操作函数。

文本操作	函数摘要
	10 进制文本至整数转换 ☞" ■ 十进制文本至整数转换 " (p21-145) 该函数用于将十进制文本转换成整数。
	十六进制文本至整数转换 ☞" ■ 十六进制文本至整数转换 " (p21-147) 该函数将十六进制文本转换成整数。
	从内部寄存器到数据缓冲器 ☞" ■ 内部寄存器到数据缓冲器 " (p21-149) 内部寄存器中保存的字符串数据被复制到数据缓冲器。
	从数据缓冲器到内部寄存器 ☞" ■ 从数据缓冲器到内部寄存器 " (p21-151) 数据缓冲器中保存的字符串数据被复制到内部寄存器。
	状态 ☞" ■ 文本操作错误状态 " (p21-153) 保存已发生的任何错误。
	数值十进制字符串转换 ☞" ■ 数值十进制字符串转换 " (p21-154) 该函数用于将整数转换成十进制字符串。
	数值十六进制字符串转换 ☞" ■ 数值十六进制字符串转换 " (p21-155) 该函数用于将二进制数据转换成十六进制字符串。
	复制部分字符串 ☞" ■ 部分文本 " (p21-156) 从字符串的指定偏移位置根据字符串长度提取数据并将其保存到另一个数据缓冲器中。
	文本设置 ☞" ■ 文本设置 " (p21-157) 固定字符串保存在数据缓冲器中。
	字符串长度 ☞" ■ 文本长度 " (p21-158) 获取已保存字符串的长度。
	字符串连接 ☞" ■ 字符串连接 " (p21-159) 用文本缓冲器将字符串或字符代码连接在一起。

■ 十进制文本至整数转换

项目	描述
摘要	该函数用于将十进制字符串转换成整数。将参数 2(转换源数据缓冲器) 中的十进制整数文本转换成整数并将它保存在参数 1(转换到地址) 中。
格式	_decasc2bin([转换到地址]、 [转换源数据缓冲器])  参数 1：　内部寄存器、临时地址 参数 2：　数据缓冲器

表达式示例 1(当数据长度为 16 位时)

_decasc2bin ([w:[#INTERNAL]LS0100], databuf0)

"databuf0" 的内容如下：

	8 位	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

上面的数据转换如下。

	16 位
LS0100	1234

表达式示例 2(当数据长度为 32 位时)

_decasc2bin ([w:[#INTERNAL]LS0100], databuf0)

"databuf0" 的内容如下：

	8 位	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	35h	'5'
databuf0[5]	36h	'6'
databuf0[6]	37h	'7'
databuf0[7]	38h	'8'
databuf0[8]	00h	NULL

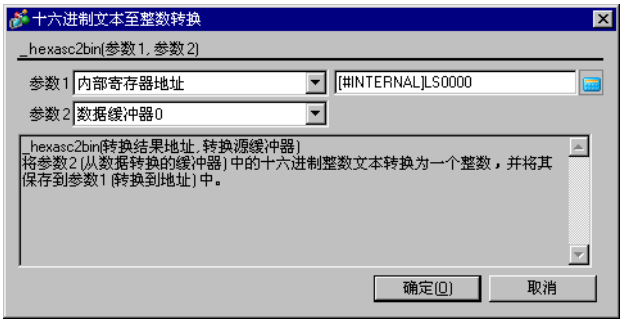
上面的数据转换如下。

	32 位
LS0100	12345678
LS0102	

重 要

- 如果已转换的位长度大于 D 脚本编辑器的位长度，会发生错误。
  例如，当脚本的位长度是 16 位时：
  _strset (databuf 0, " 123456") // 当意外设置了 6 位十进制字符串时
  _decasc2bin ([w:[#INTERNAL]LS0100], databuf0)
  当执行了上面的表达式时，字符串错误状态 [e：STR_ERR_STAT] 的错误编号 2(字符串转换错误) 但是，当发生错误时该位返回到主函数开始的地方。因此，您不能在执行了 _decasc2bin 后直接引用其他函数。(如果在函数运行时出现该命令，它就返回到调用该函数的行。)
- 在转换包含非 “0” 至 “9” 字符的数据字符串时会发生错误。
  例如，当脚本的位长度是 16 位时：
  _strset (databuf0, "12AB") // 当意外设置了非十进制字符串时
  _decasc2bin ([w:[#INTERNAL]LS0100], databuf0)
  当执行了上面的表达式时，字符串错误状态 [e：STR_ERR_STAT] 的错误编号 2(字符串转换错误) 但是，当发生错误时该位返回到主函数开始的地方。因此，您不能在执行了 _decasc2bin 后直接引用其他函数。(如果在函数运行时出现该命令，它就返回到调用该函数的行。)
- 当发生错误并返回到主函数开始的地方时，该处理终止。(如果在函数运行时出现该命令，它就返回到调用该函数的行。)

■ 十六进制文本至整数转换

项目	描述
摘要	该函数将十六进制字符串转换成二进制数据。将参数 2(转换源缓冲器) 中的十六进制整数文本转换成整数并将它保存在参数 1(转换结果地址) 中。
格式	_hexasc2bin([转换到地址]、 [转换源数据缓冲器])  参数 1：　内部寄存器、临时地址 参数 2：　数据缓冲器

表达式示例 1(当数据长度为 16 位时)

_hexasc2bin ([w:[#INTERNAL]LS0100], databuf0)

"databuf0" 的内容如下：

	8 位	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

上面的数据转换如下。

	16 位
LS0100	1234h

表达式示例 2(当数据长度为 32 位时)

_hexasc2bin ([w:[#INTERNAL]LS0100], databuf0)

"databuf0" 的内容如下：

	8 位	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	35h	'5'
databuf0[5]	36h	'6'
databuf0[6]	37h	'7'
databuf0[7]	38h	'8'
databuf0[8]	00h	NULL

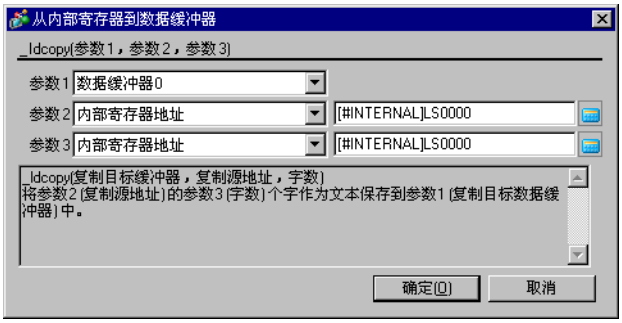
上面的数据转换如下。

	32 位
LS0100	12345678h
LS0102	

重 要

- 当转换字符串大于 16 位或 32 时会发生错误。
  例如，当脚本的位长度是 16 位时：
  _strset (databuf0, "123456")
  _hexasc2bin ([w:[#INTERNAL]LS0100], databuf0)
  当执行了上面的表达式时，字符串错误状态 [e：STR_ERR_STAT] 的错误编号 2(字符串转换错误)
- 在转换包含非 “0” 至 “9”、“A” 至 “F” 或 “a” 至 “f” 字符的数据字符串时会发生错误。
  例如，当脚本的位长度是 16 位时：
  _strset (databuf 0, "123G")
  _hexasc2bin ([w:[#INTERNAL]LS0100], databuf0)
  当执行了上面的表达式时，字符串错误状态 [e：STR_ERR_STAT] 的错误编号 2(字符串转换错误)
- 当发生错误并返回到主函数开始的地方时，该处理终止。(如果在函数运行时出现该命令，它就返回到调用该函数的行。)

■ 内部寄存器到数据缓冲器

项目	描述
摘要	根据字符串数，将保存在 LS 区的字符串数据以逐字节传输的方式复制到数据缓冲器。从参数 2(复制源地址) 将数量为参数 3(字数) 的数据作为文本复制到参数 1(复制目标数据缓冲器)。
格式	_ldcopy (复制目标数据缓冲器、 [复制源地址]、字数)  参数 1：　数据缓冲器 参数 2：　内部寄存器 参数 3：　整数值、内部寄存器、临时地址 (参数 3 的有效范围为 0 至 1024)

表达式示例 1：

_ldcopy (databuf0, [w:[#INTERNAL]LS0100], 4)

	16 位
LS0100	31h
LS0101	32h
LS0102	33h
LS0103	34h

将 LS0100 至 LS0103 中的数据从 “databuf0” 开始，顺序写入数据缓冲器的 4 字节中。按字节 (最低位) 读取 LS 区。

	8 位	
databuf0[0]	31h	‘1’
databuf0[1]	32h	‘2’
databuf0[2]	33h	‘3’
databuf0[3]	34h	‘4’
databuf0[4]	00h	NULL

重 要

- LS 区的低字节被读出，指定数量的数据被写入数据缓冲器。
- 可以为参数 3 指定的最大值是 1024。当设置了一个超出该限制的值时，字符串错误状态 [e:STR_ERR_STAT] 的错误编号 1(字符串溢出) 就被触发。
- 即使内部寄存器的上部字节中有数据，也只读来自下部字节的数据。
- 当发生错误并返回到主函数开始的地方时，该处理终止。(如果在函数运行时出现该命令，它就返回到调用该函数的行。)

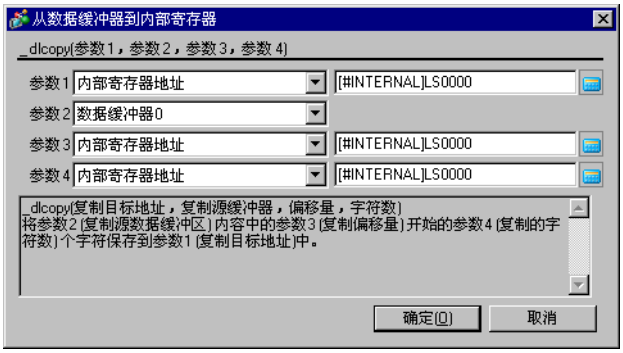
```
_ldcopy (databuf0, [w:[#INTERNAL]LS0100], 4)
```

	16 位
LS0100	3132h
LS0101	3334h
LS0102	3536h
LS0103	3738h

当如上所示保存数据时，来自下部字节的数据被读取且写入数据缓冲器中。

	8 位	
databuf0[0]	32h	‘2’
databuf0[1]	34h	‘4’
databuf0[2]	36h	‘6’
databuf0[3]	38h	‘8’
databuf0[4]	00h	NULL

■ 从数据缓冲器到内部寄存器

项目	描述
摘要	根据字符串数将保存在数据缓冲器偏移中的字符串数据的每个字节复制到 LS 区。 从参数 2(复制源数据缓冲器) 内容中的参数 3(复制源偏移值) 将参数 4(复制字符数) 个字符数据保存到参数 1(复制目标地址) 中。
格式	_dlcopy ([复制目标地址]、复制源数据缓冲器、复制源偏移值、复制字符数)  参数 1：　内部寄存器 参数 2：　数据缓冲器 参数 3：　数值、内部寄存器、临时地址 (参数 3 的有效范围在 0 至 1024 之间。) 参数 4：　数值、内部寄存器、临时地址 (参数 4 的有效范围在 1 至 1024 之间。)

表达式示例 1：

_dlcopy ([w:[#INTERNAL]LS0100], databuf0, 2, 4)

	8 位	
databuf0[0]	31h	‘1’
databuf0[1]	32h	‘2’
databuf0[2]	33h	‘3’
databuf0[3]	34h	‘4’
databuf0[4]	35h	‘5’
databuf0[5]	36h	‘6’
databuf0[6]	37h	‘7’
databuf0[7]	38h	‘8’

将 “databuf0” 的 “偏移 2” 开始提取的 4 字节数据写入 LS0100 到 LS0103。以 1 字节为单位将数据写入 LS 区。

	16 位
LS0100	33h
LS0101	34h
LS0102	35h
LS0103	36h

重 要

- 从数据缓冲器中读出 1 字节数据并将其写入 LS 区。这意味着只使用 LS 区的低 8 位 (1 字节)。高 8 位 (1 字节) 将被清 “0”。
- 当指定值 [源偏移值 + 要复制的字符数] 大于数据缓冲器大小时，字符串错误状态 [e:STR_ERR_STAT] 的错误编号 3(字符串提取错误) 将被触发。
- 当发生错误并返回到主函数开始的地方时，该处理终止。 (如果在函数运行时出现该命令，它就返回到调用该函数的行。)

■ 文本操作错误状态

当在执行文本操作的过程中发生错误时，会将错误设置到文本操作错误状态 [e: STR_ERR_STAT] 中。[e: STR_ERR_STAT] 中的“0” 表示正常状态，而保存在 [e: STR_ERR_STAT] 中的非 “0”值 表示错误状态。会将最近的错误保存在文本操作错误状态 [e: STR_ERR_STAT] 中。可以用 D 脚本工具箱菜单下的 [串口操作 / 标签设置] 设置文本运算错误状态。下表列出了文本运算错误。

错误编号	错误消息	描述
0	正常	无错误
1	文本溢出	以下函数的参数中直接包含至少 256 个字节的字符串：_strset ( ), _strlen ( ), _strcat ( ), _strmid ( ), and IO_READ_WAIT ( )。 或者，在执行 _strcat ( ) 或 _ldcopy ( ) 函数过程中创建了一个超出数据缓冲器大小的字符串。 例如： _strcat (databuf0, databuf1) 当 1020 个字节的字符串保存在 databuf0、 60 个字节的字符串保存在 databuf 1 执行上面的功能。 (超过 1024 个字节的字符串、数据缓冲器的大小导致错误状态)
2	字符串转换错误	为 _hexasc2bin ( ) 或 _decasc2bin ( ) 函数指定了无效字符代码。 例如： 在 _hexasc2bin ( ) 的第二个参数中包含了非 “0” 至 “9”、“A” 至 “F” 或 “a” 至 “f” 的字符代码。
3	字符串检索错误	试图检索一个长度大于用 “_strmid ( )” 函数指定的字符串长度的字符串。或者分配了大于指定字符串的偏移值。 例如： _strmid (databuf0, "12345678", 2, 8) 试图从偏移 2 中检索一个 8 字符的字符串。

D 脚本和全局 D 脚本不能使用字符串控制错误状态。如果它被意外读取，将载入 “0”。

在执行每个函数时都将其保存在错误状态中。

要检查错误状态 [e: STR_ERR_STAT]，请写如下语句。您可以用如下表达式确认错误。

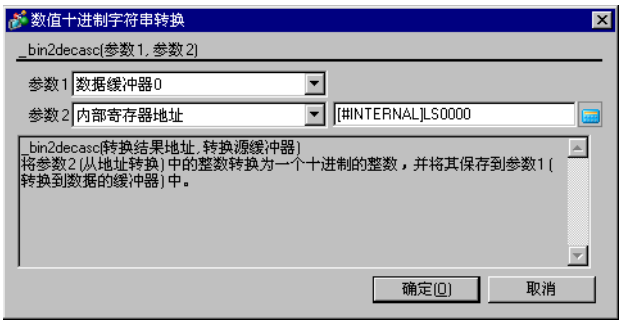
表达式示例：

```
if ([e:STR_ERR_STAT] <> 0)          // 检查错误状态。
{
    set([b:[#INTERNAL]LS005000])    // 在错误显示指示灯上置位
}
endif
```

重 要

- 当发生错误并返回到主函数开始的地方时，该处理终止。 (如果在函数运行时出现该命令，它就返回到调用该函数的行。)

■ 数值十进制字符串转换

项目	描述
摘要	该函数用于将整数转换成十进制字符串。将参数 2(转换源地址) 中的整数转换成十进制整数文本并将其保存在参数 1(转换到数据缓冲器) 中。
格式	_bin2decasc(转换结果地址 , 转换源缓冲器)  参数 1：　数据缓冲器 参数 2：　内部寄存器、临时地址

表达式示例 1(当数据长度为 16 位时)

_bin2decasc (databuf0, [w:[#INTERNAL]LS0100])

	16 位
LS0100	1234

上面的数据转换如下。注意增加了 “NULL(0x00)”。

	8 位	
databuf0[0]	31h	‘1’
databuf0[1]	32h	‘2’
databuf0[2]	33h	‘3’
databuf0[3]	34h	‘4’
databuf0[4]	00h	NULL

表达式示例 2(当数据长度为 32 位时)

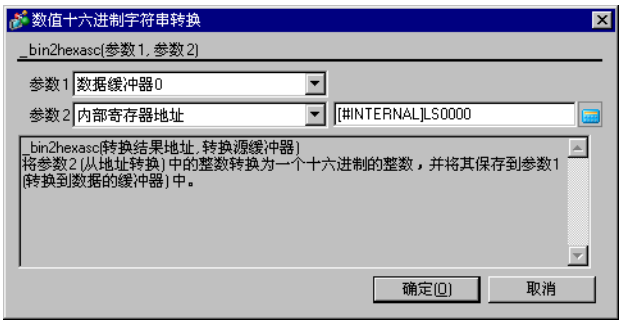
_bin2decasc (databuf0, [w:[#INTERNAL]LS0100])

	32 位
LS0100	12345678
LS0102	

上面的数据转换如下。

	8 位	
databuf0[0]	31h	‘1’
databuf0[1]	32h	‘2’
databuf0[2]	33h	‘3’
databuf0[3]	34h	‘4’
databuf0[4]	35h	‘5’
databuf0[5]	36h	‘6’
databuf0[6]	37h	‘7’
databuf0[7]	38h	‘8’
databuf0[8]	00h	NULL

■ 数值十六进制字符串转换

项目	描述
摘要	该函数用于将二进制数据转换成十六进制字符串。将参数 2(转换源地址) 中的整数转换成十六进制整数文本并将其保存在参数 1(转换到数据缓冲器) 中。
格式	_bin2hexasc(转换到数据缓冲器、 [转换源地址])  参数 1：　数据缓冲器 参数 2：　内部寄存器、临时地址

表达式示例 1(当数据长度为 16 位时)

_bin2hexasc (databuf0, [w:[#INTERNAL]LS0100])

	16 位
LS0100	1234h

上面的数据转换如下。注意增加了 “NULL(0x00)”。

	8 位	
databuf0[0]	31h	‘1’
databuf0[1]	32h	‘2’
databuf0[2]	33h	‘3’
databuf0[3]	34h	‘4’
databuf0[4]	00h	NULL

表达式示例 2(当数据长度为 32 位时)

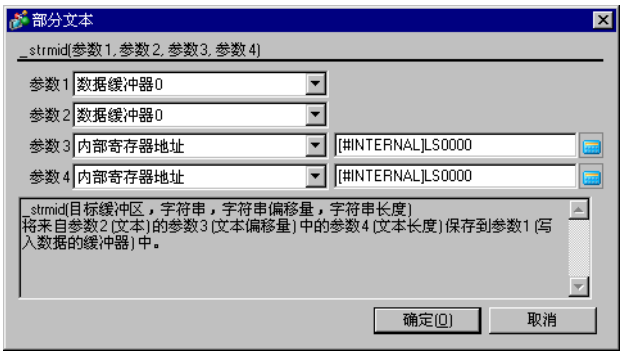
_bin2hexasc (databuf0, [w:[#INTERNAL]LS0100])

	32 位
LS0100	12345678h
LS0102	

上面的数据转换如下。

	8 位	
databuf0[0]	31h	‘1’
databuf0[1]	32h	‘2’
databuf0[2]	33h	‘3’
databuf0[3]	34h	‘4’
databuf0[4]	35h	‘5’
databuf0[5]	36h	‘6’
databuf0[6]	37h	‘7’
databuf0[7]	38h	‘8’
databuf0[8]	00h	NULL

■ 部分文本

项目	描述
摘要	从字符串的指定偏移位置根据字符串长度提取数据并将其保存到另一个数据缓冲器中。从参数 2(文本) 的参数 3(文本偏移) 开始将长度为参数 4(文本长度) 的文本保存到参数 1(写入数据缓冲器) 中。
格式	_strmid (写入到数据缓冲器，文本，文本偏移，文本长度)  参数 1：　数据缓冲器 参数 2：　字符串、数据缓冲器 参数 3：　数值、内部寄存器、临时地址 (参数 3 的有效范围为 0 至 1024) 参数 4：　数值、内部寄存器、临时地址 (参数 4 的有效范围为 1 至 1024)

表达式示例：

_strmid (databuf0, "12345678", 2, 4)

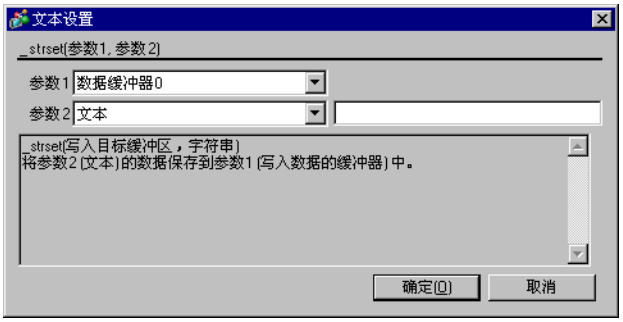
从字符串 “12345678”的偏移 2 开始检索 4 字节数据保存到 “databuf0”中。

	8 位	
databuf0[0]	33h	‘3’
databuf0[1]	34h	‘4’
databuf0[2]	35h	‘5’
databuf0[3]	36h	‘6’
databuf0[4]	00h	NULL

重 要

- 当试图检索一个长度大于 “strmid ( )”函数指定的字符串长度的字符串时，或当指定了一个大于指定字符串的偏移值时，字符串错误状态 [e:STR_ERR_STAT] 的错误编号 3(字符串提取错误) 将被触发。
- 当发生错误并返回到主函数开始的地方时，该处理终止。(如果在函数运行时出现该命令，它就返回到调用该函数的行。)

■ 文本设置

项目	描述
摘要	固定字符串保存在数据缓冲器中。将参数 2(文本) 的数据保存到参数 1(写入数据缓冲器) 中。
格式	_strset(写入目标缓冲器 , 字符串)  参数 1：　数据缓冲器 参数 2：　文本、数值 (文本代码)(参数 2 的有效范围在 1 至 255 之间。)

表达式示例：

_strset (databuf0, "ABCD")

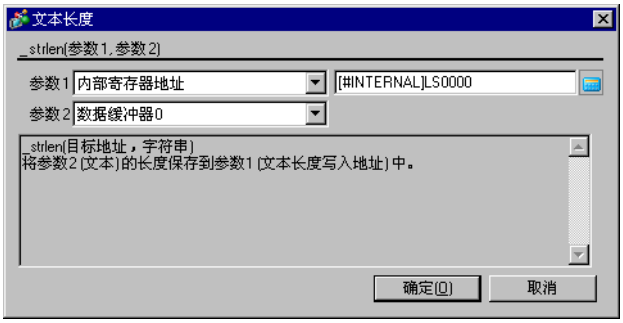
字符串如下所示保存在数据缓冲器中的：

	8 位	
databuf0[0]	41h	'A'
databuf0[1]	42h	'B'
databuf0[2]	43h	'C'
databuf0[3]	44h	'D'
databuf0[4]	00h	NULL

重 要

- 最多可以指定具有 255 个字符的字符串。要创建长于该限定值的字符串，可将多出的字符串保存在另外一个缓冲器中，然后用字符串连接函数 (_strcat) 将两者连接起来。
- 要清空数据缓冲器，请创建一个空字符串。
  例如：_strset (databuf0,"")
  　　_strset (databuf0,0)

■ 文本长度

项目	描述
摘要	获取已保存字符串的长度。将参数 2(文本) 的长度保存在参数 1(文本长度写入地址) 中。 (不包括 NULL 字符。)
格式	_strlen(目标地址 , 字符串)  参数 1：　内部寄存器、临时地址 参数 2：　字符串、数据缓冲器

表达式示例 1：

_strlen ([w:[#INTERNAL]LS0100], "ABCD")

当执行了上述语句时，会将字符串长度按如下所示写入 LS0100。

LS0100	4

表达式示例 2：

_strlen ([t:0000], databuf0)

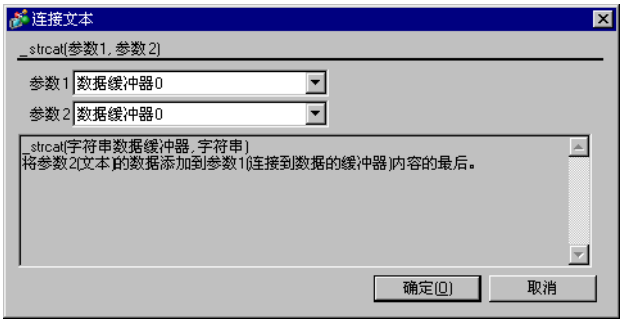
"databuf0" 的内容如下：

	8 位	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

当执行了上述语句时，会将字符串长度按如下所示 写入 [t:0000]。

t0000	4

■ 字符串连接

项目	描述
摘要	用文本缓冲器将字符串或字符代码连接在一起。将参数 2(文本) 的数据添加到参数 1(连接到数据的缓冲器) 内容的后面。
格式	_strcat(字符串数据缓冲器 , 字符串)  参数 1：　数据缓冲器 参数 2：　文本，数值 (文本代码)，数据缓冲器 (参数 2 的有效范围在 1 至 255 之间。)

表达式示例 1：

_strcat (databuf0, "ABCD")

	8 位	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

当如上所述连接 "ABCD" 时，结果如下。注意增加了 "NULL(0x00)"。

	8 位	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	41h	'A'
databuf0[5]	42h	'B'
databuf0[6]	43h	'C'
databuf0[7]	44h	'D'
databuf0[8]	00h	NULL

重 要

- 最多可以指定具有 255 个字符的字符串。
- 如果为参数 2 设置了数值 0 或空字符串，参数 1 的数据缓冲器不会改变。例如：_strcat (databuf0,"")
  _strcat (databuf0,0)

21.11.12 运算示例

■ 逻辑运算示例

下面显示逻辑运算示例。

◆ **((100 > 99) and (200 <> 100))**

结果：ON

◆ **((100 > 99) and (200 <> 200))**

结果：OFF

◆ **((100 > 99) or (200 <> 200))**

结果：ON

◆ **((100 < 99) or (200 <> 200))**

结果：OFF

◆ **not (100 > 99)**

结果：OFF

◆ **not (100 < 99)**

结果：ON

◆ **[w:[PLC1]D200] < 10**

结果：如果 D200 小于 10 时成立。

◆ **not [w:[PLC1]D200]**

结果：如果 D200 等于 0 时成立。

◆ **([w:[PLC1]D200] == 2) 或 ([w:[PLC1]D200] == 5)**

结果：如果 D200 等于 2 或 5 时成立。

◆ **([w:[PLC1]D200] < 5) 和 ([w:[PLC1]D300] < 8)**

结果：如果 D200 小于 5，且 D300 小于 8 时成立。

◆ **[w:[PLC1]D200] < 10**

结果：如果 D200 小于 10 时成立。

◆ **not [w:[PLC1]D200]**

结果：如果 D200 等于 0 时成立。

◆ **([w:[PLC1]D200] == 2) 或 ([w:[PLC1]D200] == 5)**

结果：如果 D200 等于 2 或 5 时成立。

◆ **([w:[PLC1]D200] < 5) 和 ([w:[PLC1]D300] < 8)**

结果：如果 D200 小于 5，且 D300 小于 8 时成立。

■ 位运算示例

下面给出一些位运算示例。

◆ [w:[PLC1]D200] << 4

结果：D200 中的数据向左移 4 位。

◆ [w:[PLC1]D200] >> 4

结果：D200 中的数据向右移 4 位。

◆ 将 12(0000Ch) 以二进制形式保存到 D301 中。

[w:[PLC1]D200] = [w:[PLC1]D300] >> [w:[PLC1]D301]

结果：D300 中的数据向右移 12 位并赋值给 D200。

◆ [w:[PLC1]D200] << 4

结果：D200 中的数据向左移 4 位。

◆ [w:[PLC1]D200] >> 4

结果：D200 中的数据向右移 4 位。

◆ 将 12(0000Ch) 以二进制形式保存到 D310 中。

[w:[PLC1]D200] = [w:[PLC1]D300] >> [w:[PLC1]D310]

结果：D300 中的数据向右移 12 位并赋值给 D200。

◆ 按位与

0 & 0	结果 : 0
0 & 1	结果 : 0
1 & 1	结果 : 1
0x1234 & 0xF0F0	结果 : 0x1030

◆ 按位或

0 | 0	结果 : 0
0 | 1	结果 : 1
1 | 1	结果 : 1
0x1234 | 0x9999	结果 : 0x9BBD

◆ 按位异或

0 ^ 0	结果 : 0
0 ^ 1	结果 : 1
1 ^ 1	结果 : 0

◆ 按位取反 (当数据格式是 BIN16+)

~ 0	结果 : 0xFFFF
~ 1	结果 : 0xFFFE

■ 使用条件分支的计算示例

使用 "if-endif" 和 "if-else-endif" 的控制程序流

◆ if-endif

```
if ( 条件 )
{Process 1}
endif
```

如果条件成立， Process 1 运行。如果不成立，跳过 Process 1。

例如：

```
if ( [ w:[PLC1]D200 ] < 5 )
{
    [ w:[PLC1]D100 ] = 1
}
endif
```

如果 D200 中的数据小于 5，则将 1 赋值给 D100。

◆ if-else-endif

```
if ( 条件 )
{Process 1}
else
{Process 2}
endif
```

如果条件成立，运行 Process 1。如果不成立，运行 Process 2。

例如：

```
if ( [ w:[PLC1]D200 ] < 5 )
{
    [ w:[PLC1]D100 ] = 1
}
else
{
    [ w:[PLC1]D100 ] = 0
}
endif
```

如果 D200 中的数据小于 5，则将 1 赋值给 D100。否则，赋值 0。

■ 使用偏移地址的计算示例

偏移指定：特殊计算示例，使用 [w:D00100]#[t:0000]。

◆ 脚本 I/O：16 位无符号， [t:0000]= 65526，结果地址是 [w:[PLC1]D00090]。

100 + 65526 = 64(Hex) + FFF6(Hex) = 1005A(Hex) --> 005A(Hex) = 90

|

低 16 位有效

◆ 脚本 I/O：16 位有符号， [t:0000]= -10，结果地址是 [w:[PLC1]D00090]。

100 + (-10) = 64(Hex) + FFF6(Hex) = 1005A(Hex) --> 005A(Hex) = 90

|

低 16 位有效

◆ 脚本 I/O：32 位无符号， [t:0000]= 4294901840，结果地址是 [w:[PLC1]D00180]。

100 + 4294901840 = 64(Hex) + FFFF0050(Hex) = FFFF 00B4(Hex) --> 00B4(Hex) = 180

|

低 16 位有效

◆ 脚本 I/O：32 位有符号， [t:0000]= -65456，结果地址是 [w:[PLC1]D00180]。

100 + (-65456) = 64(Hex) + FFFF0050(Hex) = FFFF 00B4(Hex) --> 00B4(Hex) = 180

|

低 16 位有效

重要

- 无论脚本的位长度和数据类型如何设置，总是将偏移地址按 16 位二进制值处理。如果结果超过 16 位 (最大值：65535)，将位 0 至位 15 视为有效位，忽略位 16 及以上的位。